QUASAR: A Method for the QUality Assessment of Software-Intensive System ARchitectures

Donald Firesmith, Software Engineering Institute

In Collaboration with
Peter Capell, Software Engineering Institute
Joseph P. Elm, Software Engineering Institute
Michael Gagliardi, Software Engineering Institute
Tim Morrow, Software Engineering Institute
Linda Roush, Naval Air Systems Command, U.S. Navy
Lui Sha, University of Illinois at Urbana-Champaign

July 2006

HANDBOOK
CMU/SEI-2006-HB-001

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

QUASAR: A Method for the QUality Assessment of Software-Intensive System ARchitectures

CMU/SEI-2006-HB-001

Donald Firesmith, Software Engineering Institute

In Collaboration with
Peter Capell, Software Engineering Institute
Joseph P. Elm, Software Engineering Institute
Michael Gagliardi, Software Engineering Institute
Tim Morrow, Software Engineering Institute
Linda Roush, Naval Air Systems Command, U.S. Navy
Lui Sha, University of Illinois at Urbana-Champaign

July 2006

Acquisition Support Program

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Acknowledgements

We would like to acknowledge and thank the following organizations and people on whose work the QUality Assessment of System ARchitectures (QUASAR) method has been largely based, either directly or indirectly, over the last three years:

- Joint Strike Fighter (JSF) Program Office
- Lockheed Martin JSF Architects
- Safety Case Developers including safety engineers at Adelard LLP

We would like to acknowledge the following individuals from the Software Engineering Institute (SEI) who supported the development of this handbook:

- Rick Barbour, Chief Engineer, Navy, who provided administrative management for SEI support to the JSF Joint Program Office (JPO)
- Brian Gallagher, Director, Acquisition Support Program (ASP), who provided administrative leadership for the development of this handbook
- John Goodenough, Chuck Weinstock, and John Hudak, whose work on using the Goal Structuring Notation to document assurance cases emphasizes the importance of graphically summarizing quality cases and organizing claims, arguments, and evidence around individual quality cases [Weinstock 04]
- Susan Kushner, who did a great job editing this handbook

We would like to acknowledge the following SEI individuals who reviewed the draft handbook and provided numerous useful comments and recommendations:

- Ron Kohl (SEI Visiting Scientist)
- Harry Levinson
- Tom Merendino, especially with regard to method tailoring
- Mike Phillips

Executive Summary

The quality of a system's architecture is critical to that system's success. This is especially true for software-intensive systems, which often have very complex system and software architectures. Thus, a system's ultimate success depends on how well its architecture helps it to meet its architecturally significant requirements. The quality of the overall system architecture also depends on the quality of the architectures of the system's subsystems, the quality of the architectures of their subsystems, and so on. Unless the architectures of these subsystems and sub-subsystems adequately help them meet the derived architecturally significant requirements that are allocated to them, it is unlikely that the overall quality of the system architecture will be adequate. Without this proper foundation, it becomes very difficult and expensive to achieve sufficient system quality during design, implementation, and testing.

Unlike modern software that is often organized along object lines, systems typically continue to be decomposed functionally into subsystems. System architectures therefore tend to be driven by cohesive groupings of functional requirements (i.e., feature sets). But frequently, the system's architecture should be driven as much or more by its *quality* requirements as by its functional requirements. In other words, it is important for the system architecture to help ensure that the system achieves *sufficient* levels of important quality factors, such as affordability, availability, capacity, correctness, efficiency, interoperability, modifiability, performance, portability, producibility, reliability, reusability, robustness, safety, scalability, security, sustainability, testability, and usability. These quality factors often become the basis of the most important types of architecturally significant requirements: the quality requirements. Unfortunately because quality requirements are often poorly specified and legitimate stakeholder needs may not be specified at all, there is a significant risk that a system's architecture will fail to adequately support its true quality requirements.

It is very important to ensure that a system's quality requirements be properly derived and allocated to its subsystems and their subsystems. It is also important to assess the quality of the architectures of the system and its subsystems to ensure that these architectures will sufficiently enable subsystems to meet the derived quality requirements that are allocated to them.

This handbook documents the QUASAR (QUality Assessment of System ARchitectures) system architecture quality assessment method, which is a practical method for assessing the quality of *system* architectures in terms of the degree to which the architectures of their subsystems and their sub-subsystems help ensure that they meet the derived quality requirements allocated to them. This handbook documents the QUASAR method in terms of the

1. challenges the method was developed to meet

2. objectives of the method as well as its philosophy and assumptions
3. concept of quality cases on which the method has been based
4. method's major phases and component tasks
5. makeup and responsibilities of the teams that collaborate to perform these tasks
6. resulting work products that the teams produce when performing these tasks
7. lessons learned when performing earlier versions of the QUASAR method while assessing the quality of the architecture of a software-intensive system of systems
8. appendices defining terms, providing reusable checklists, and giving examples of quality cases

QUASAR is based on the premise that the system architects are responsible for

- knowing and understanding the relevant derived and allocated goals and requirements that their architectures must help their subsystems fulfill
- creating an appropriate architecture that supports the meeting of these requirements
- properly documenting this architecture so that their architectural decisions and associated rationales can be readily found
- knowing whether their architectures sufficiently support the requirements that have been allocated to them
- therefore, being able to make a strong case that their architectures have sufficient quality

Based on a generalization of the idea of a *safety case*, which is widely used in the safety community, QUASAR is a structured way for the architecture team to convince the assessment team that the architecture has adequate quality and for the assessment team to determine the veracity of the architecture team's claims. For each important quality factor or quality subfactor, the architecture team makes an associated quality case that their architecture meets associated derived and allocated requirements. Thus, the architects could present an extensibility case, interoperability case, performance case, reliability case, and safety case during an assessment. Each such quality case consists of the following information:

- *claims* that the architecture adequately helps the system achieve its associated quality goals and meet its quality requirements
- clear and compelling *arguments* (in terms of cohesive sets of architectural decisions and their associated rationales) that justify belief in these claims
- sufficient *evidence* (e.g., official project architecture diagrams, models, and documents as well as any witnessed demonstrations) to support the architects' arguments

After discussing the tasks and steps of the method, the teams that perform these tasks, and the work products that they produce, this handbook includes a list of lessons learned during actual usage on a very large, complex program. The handbook concludes with appendices con-

taining definitions of quality factors and their subfactors, reusable checklists, and simplified and sanitized examples of claims, arguments, and evidence that architects might provide during assessments.

Abstract

This handbook documents the QUASAR (QUality Assessment of System ARchitectures) method for assessing the quality of the architecture of a software-intensive system. It begins by discussing the challenges that are faced when assessing a system's architecture and outlines the development history of the method. The next section of the handbook documents the concept of quality cases and the claims, arguments, and evidence that compose them. This is followed by a description of the teams that collaborate to perform QUASAR tasks. Next, individual tasks and associated steps performed as part of the QUASAR method are documented. Next, the work products produced by these teams when performing these tasks are described. Finally, lessons learned during the development and use of the method when assessing the quality of major subsystems during the development of a very large, software-intensive system of systems are presented. Also provided are appendices that define common quality factors and subfactors, offer reusable checklists, and give examples of quality cases. The example quality cases illustrate valid quality goals and requirements that compose claims, example architecture decisions and associated rationales that compose arguments, and the types of evidence that architects might provide.

1 Introduction

This section of the handbook documents the goals of the QUASAR *system architecture* quality assessment method and the challenges that have led to its creation. It also provides a brief history of the development and verification of the assessment method.

1.1 Intended Audiences

This handbook is intended for anyone who may mandate or take part in a system architecture quality assessment. This includes, but is not limited to, people who fill one or more of the following roles:

- **System Acquisition (Customer) Personnel**
  - **Customer**
    Someone who may contractually mandate the performance of such assessments
- **Assessment Team Member**
  - **Assessor**
    Someone who is responsible for technically assessing the quality of the system architecture against its quality requirements in terms of the architectural information provided by the architects
- **Subject Matter Expert**

  Someone who acts as an expert during system architecture quality assessments
- **System Development Personnel**
  - **Manager**
    Someone who mandates that development staff perform the assessments, either internally within the development organization or externally in cooperation with members of the acquisition organization
  - **Requirements Engineer**
    Someone who is a member of the requirements team and who develops and presents architecturally significant goals and requirements during the subsystem requirements review part of the assessment
  - **Architect**
    Someone who is a member of the architecture team, who develops the system and subsystem architectures, and who develops and presents the quality cases to the assessment team

– **Trainer**
Someone who provides training in how to perform system architecture quality assessments

1.2 Goals of this Handbook

This handbook has two primary goals.

1. **Enable method use.**

   This handbook is intended to enable readers to understand and use the QUASAR method. It clearly documents all of the major components of QUASAR method including its phases and their component tasks, the steps making up these tasks, the participating teams and roles people play as members of these teams, and the work products they produce. This handbook captures the best practices in system architecture assessment in a single, convenient source of ready information that will help practitioners prepare for and perform assessments.

   Note that this is a handbook. As such, this document addresses each individual task and work product in some detail so that the reader need only read sections of the handbook that are relevant to the work at hand, rather than read the entire document. One unfortunate side effect is that individual subsection completeness results in some redundancy between highly related subsections, which may prove annoying to anyone attempting to read the handbook from cover to cover. Readers falling into the latter category can feel free to skim over any duplicate subsections without fear of missing critical information.

2. **Provide justification for use.**

   Because performing effective system architecture quality assessments requires a significant expenditure of resources in terms of effort expended by some of a project's most critical staff, the handbook also provides sufficient business and technical reasons to justify the method's introduction and use.

1.2.1 Properly Document QUASAR

The primary purpose of this handbook is to properly document the QUASAR method in sufficient detail so that it can be introduced and used on system development programs.

This handbook begins by discussing the challenges that are faced when assessing the quality of a system's architecture in terms of the architecture's support for achieving its allocated architecturally significant requirements. This handbook also outlines the development and verification history of the QUASAR method. Section 2 gives an overview of the QUASAR method including its objectives, philosophy, and the assumptions on which it is built. Section 3 introduces the reader to the concept of quality cases. Section 4 of the handbook documents the individual tasks and associated steps that are performed as a part of the QUASAR method. Section 5 documents the three teams that collaborate to perform the QUASAR tasks. Section 6 describes of the work products that are produced by teams when performing the

assessment tasks. Section 7 lists the lessons that have been learned during the development and use of the method when assessing the quality of the architectures of major subsystems during the development of a very large system of systems. Finally, the appendices define commonly used quality factors and subfactors, provide reusable checklists, and most importantly, provide the reader with examples of the kinds of information typically composing the quality cases that architects develop and provide to the assessors.

1.2.2 Justify Use of the QUASAR Method

On many systems development programs, the system architects tend to be considered some of the most important members of the technical development staff. Not only is the architecture critical to the success of the system, the lead architect often ends up being the technical leader on small and medium-sized programs who assumes all of the other technical and managerial responsibilities that the position brings. Similarly, other lower level architects tend to be the leaders of their subsystem integrated project teams (IPTs). Given that most system development programs tend to have a shorter schedule than optimal, the system architects tend to be extremely busy developing the system architectures, communicating these architectures with stakeholders, and ensuring the integrity of the architectures as they are flowed down into the lower level architectures, designs, and implementations. This is why it is important to understand and remember that architecture assessment brings significant value and return on investment to the program by significantly lowering program risks to system quality, development cost, and schedule due to inadequate system architectures. If the system architects are properly performing their jobs, then they will have properly documented their architectures as they go. At any point in time, a system architecture quality assessment should be able to be held with only minimal additional preparation time required of the architects.

Another goal of this handbook is to clearly show why you should perform an appropriate number of system architecture quality assessments when developing any software-intensive system. We are therefore obliged to provide adequate business and technical reasons to justify its use. Section 2.1 of this handbook begins with a list of QUASAR's objectives that you can use to justify expending the cost and valuable resources that using the method entails.

1.3 Challenges

The following are important challenges driving the development of the QUASAR method:

- **No system architecture quality assessment methods exist.**

  Although well-known software architecture assessment methods exist [Clements 02], no well-known, industry-standard method for evaluating system architectures against their architecturally significant requirements exists.

- **Assessments are not mandated and are considered "scope creep."**

Unfortunately, acquisition organizations typically do not require system architecture quality assessments in the acquisition contract. Unless the contract clearly specifies the performance of system architecture quality assessments, the development organization can legitimately complain that such assessments are out of the scope of the contract and would therefore require unfunded effort and schedule slippage. The acquisition organization is then faced with either renegotiating the contract with non-trivial cost and schedule increases or using their limited bag of carrots (e.g., award fees) and sticks (e.g., not approving architecture documents) as a means to force the development organization to participate in a series of system architecture quality assessments. If the contract is not renegotiated, there will be extreme pressure to limit the scope of the assessments, potentially to the point where major parts of the system architecture are either not assessed or inadequately assessed.

- **Architecturally significant requirements are poorly specified.**

  Many projects do not produce sufficient, well-specified architecturally significant requirements to drive the development of the system architecture and therefore, criteria against which to assess it. This is especially true of quality requirements (e.g., interoperability, modifiability, performance, portability, safety, security, usability). Without unambiguously measurable quality requirements, it is difficult to determine if the architecture is of *sufficient* quality.[1]

  Unless the contract clearly specifies the customer's architecturally significant requirements, it is highly unlikely that adequate architecturally significant subsystem requirements will be derived, allocated to individual subsystems, and properly specified. Without properly specified architecturally significant requirements, it is highly unlikely that the architect will have adequately guessed their existence and incorporated them into the architecture. The architect can then legitimately argue that being forced to support an unmandated, unscheduled, and unfunded system architecture quality assessment against unspecified requirements is a clear example of inappropriate scope creep. The result can be a very unproductive ring of finger pointing.

- **System architectures are inadequately documented.**

  System architects do not typically document their architectures to the degree needed to perform a proper quality assessment. Specifically, the architectural decisions supporting the fulfillment of architecturally significant requirements are often not well documented, their rationales are often missing, and traceability of these decisions back to their quality requirements is often missing (especially when these requirements themselves are poorly specified). Also, engineering tradeoff decisions to ensure sufficient support for conflict-

[1] Note that the QUASAR method *does not* include guidance for the actual engineering of quality goals and requirements. Nevertheless, because of this challenge, QUASAR *does* include a phase, one of the primary objectives of which is to ensure that such goals and requirements have been properly derived and allocated to the subsystems.

ing quality factors (e.g., interoperability and performance vs. security) are rarely adequately documented.

- **Architects are inadequately prepared for assessments.**

  Because system architecture quality assessments are often not mandated in the acquisition (development) contract, architects rarely allocate adequate resources (e.g., budget and schedule) for preparing for and participating in the assessments. Because their architectures are often poorly documented with regard to how their architectures support the meeting of their allocated quality requirements, architects typically need to produce the documentation in order to pass the assessments.

- **Architecture size and complexity can be overwhelming.**

  Modern software-intensive systems often consist of a very large hierarchy of large and complex subsystems, sub-subsystems, and so on. These subsystems may well collaborate in highly sophisticated manners to implement thousands of functional, data, interface and quality requirements. This level of size and complexity can easily overwhelm the human capacity of individual architects or even teams of architects to comprehend. Because architectural defects (e.g., mistakes, inconsistencies, and incompleteness) are inevitable, projects need an adequate number of architectural assessments to identify these defects and minimize their associated risks.

- **Incremental, parallel assessments are necessary.**

  The large size of many modern systems makes it impossible to assess their architectures all at once. When using modern iterative, incremental, parallel, and time-boxed development cycles, the system architecture also tends to be developed incrementally as its subsystems are identified and architected. This leads to the existence of multiple subsystem architecture teams working in parallel. Thus, an architecture quality assessment method usually needs to be able to incrementally assess the system's architecture as it is developed, subsystem by subsystem. Given the large number of subsystem assessments that need to occur, it may well be that multiple simultaneous subsystem assessments must occur in parallel.

  However, the assessment method also needs to be able to be scaled down for assessing smaller systems where only a single system assessment occurs.

- **Results summarization is necessary.**

  Because the system architecture quality assessment may involve the assessment of the architectures of many subsystems at multiple tiers within the aggregation hierarchy of the overall architecture, the architecture assessment method needs to be able to summarize the results of the individual subsystem architecture assessments into an overall assessment of the entire system architecture.

- **Assessments must balance architect workload with effectiveness.**

  System architects and assessors are extremely busy. Thus, the architecture assessment method should result in an appropriate balance between minimizing their workload and

maximizing assessment effectiveness in terms of identifying architecture defects and risks in order to improve the quality of the resulting architecture.

- **An experienced assessment team is needed.**

  System architectures are often very complex and highly technical, requiring experience and training in application domains (e.g., avionics and sensors) and specialty engineering (e.g., reliability, safety, and security) in order to adequately understand and assess their technical ramifications to the architecture.

- **The assessment method must be repeatable.**

  The system architecture assessment method typically needs to be repeated to assess the architectures of many different individual subsystems (or parts of subsystems) as well as the architecture of the same subsystems as they are iteratively and incrementally developed over time.

- **There is a lack of acquisition guidance regarding contract language.**

  There is currently a lack of guidance for the acquisition manager regarding appropriate content to put into the request for proposal (RFP) or contract mandating the assessment of system architectures against their required quality characteristics.

1.4 History of Development and Use

The QUASAR method was originated during the assessment of the architectures of the major subsystems of the U.S. Department of Defense (DoD) F-35 Joint Strike Fighter (JSF) aircraft system of systems.[2] Earlier versions of this method were used during a series of assessments of the architectures of both embedded aircraft systems (e.g., mission systems and vehicle systems) as well as ground-based systems (e.g., information systems and training systems). As a result of each individual assessment, substantial lessons were learned and incorporated into the method.

The QUASAR method has also been significantly based on the experience of the handbook's authors as architects and architecture assessors. It is also important to note that the specific challenges listed that justify the performance of QUASAR assessments as well as the example quality cases in the appendix of this handbook are very general, highly sanitized, and *not* related to any specific system development program.

[2] Because the SEI became involved after contract award, the SEI was neither able to ensure that system architecture quality assessments were written into the contract, nor ensure a proper early emphasis on the engineering of the quality requirements on which the QUASAR method is based. Instead, the use of system architecture quality assessments to address system compliance with a small number of contractual architecture requirements is largely due to the farsightedness of members of the acquisition staff responsible for system architecture.

2 QUASAR Overview

When following the QUASAR method, one or more assessment teams assess the quality of a system's architecture by means of quality cases, which are developed and presented to the assessment teams by the system and subsystem architecture teams. The architecture teams use these quality cases to make the case that their architectures sufficiently support the system's ability to meet its associated quality requirements. Thus, QUASAR enables the architecture teams to convince the assessment teams that their architectures provide sufficient support for necessary system quality factors.[3]

Note that QUASAR is *not* a means for assessing the architects' plans and procedures for developing their system architecture. Although this information tends to naturally become clear during QUASAR assessments, the QUASAR method neither assesses the quality or appropriateness of the system architecture process, nor does it assess whether the architecture teams are following their architecture methods. In other words, QUASAR assesses neither process goodness nor method compliance. Instead, QUASAR assesses the *actual* quality of the system architecture at one or more specific points in time.[4]

2.1 Objectives of System Architecture Quality Assessments

Understanding the many objectives for performing system architecture assessments in general and QUASAR assessments in particular, provides a strong business and technical case for investing the significant resources required to perform them. Because different system stakeholders can use architecture quality assessments to achieve different objectives, the reader is invited to select the specific, appropriate objectives from the following list when arguing for the incorporation of system architecture quality assessments into a project.

[3] Quality factors (sometimes called quality attributes and quality characteristics) include affordability, availability, capacity, configurability, correctness, efficiency, extensibility, interoperability, maintainability, modifiability, portability, producibility, reliability, reusability, robustness, safety, scalability, security, stability, sustainability, testability, and usability.

[4] The system architecture decomposes the system into its subsystems, their sub-subsystems, and so on. The system architecture is therefore assessed in terms of the architectures of its subsystems; the individual subsystem-specific assessments will naturally occur on different dates.

The QUASAR method is used to assess the quality of the architecture of systems and their subsystems in order to

1. **Determine system architecture quality.**

   The architecture of a system significantly constrains the downstream system design and implementation as well as the performance of the system integration, testing, and production activities. Because the levels of a system's quality factors are largely enabled (or made difficult to achieve) by the system's architecture, the quality of a system's architecture greatly influences the quality of the resulting system. Thus, the *primary* objective of QUASAR assessments is to determine the quality of the system's architecture in terms of the degree to which the architecture enables the system to meet its associated quality goals and requirements.

2. **Determine contract compliance.**

   Large systems are often developed by means of contracts between an acquisition (customer) organization and the development organization (also called a supplier or vendor). If the customer mandates appropriate contractually binding, quality-relevant requirements[5] as part of the acquisition contract, then to minimize project risk and exercise acquisition oversight due diligence, the acquisition organization may also mandate a series of system architecture quality assessments to verify that the development organization's complies with these contractual requirements. On such programs, a major objective of QUASAR assessments is to determine compliance of the system architecture with these architecturally significant, contractual requirements.

3. **Ensure specification of architecturally significant requirements.**

   The architecturally significant requirements are often very poorly engineered and this is especially true of the quality requirements. Many are never specified at all, or else they are incorrectly specified as ambiguous, infeasible, and unverifiable goals such as "the system shall be reliable" or "the system shall be safe." Although there are many reasons why this occurs in practice,[6] the result is the same. The architects have to guess at the required qualities of the architecture without knowing *how good is good enough* or how best to perform engineering tradeoffs between conflicting qualities such as security versus maintainability and testability. It is often not until relatively late in the program, for

[5] The primary type of quality-relevant requirements is quality requirements that specify a minimum acceptable level of some quality factor or quality subfactor. The three other types of quality-relevant requirements are (1) quality-significant requirements (e.g., functional requirements that have quality ramifications), (2) quality subsystem requirements (i.e., requirements for a quality subsystem such as a safety subsystem), and (3) quality constraints (e.g., architecture constraints that mandate a specific quality architectural mechanism, design constraints that mandate a quality design decision, or implementation constraints that mandate a quality implementation decision such as the use of a safe subset of a programming language) [Firesmith 03].

[6] The most notable of these is the myth that quality requirements cannot be unambiguously specified, which itself is due largely to a lack of training in how to properly specify such requirements.

example, during system testing, that addressing this problem becomes unavoidable. Thus, a major objective of QUASAR assessments is to ensure that the architecture can be successfully assessed. In order to assess the architecture against its support for the architecturally significant quality requirements, these requirements must exist, they must have the proper characteristics (e.g., be verifiable and unambiguous), and they must have been properly specified sufficiently early in the development process to drive the development of the architecture.

Note however that as a system *architecture* quality assessment method, QUASAR does *not* include the requirements engineering tasks during which the architecturally significant quality goals and requirements are derived and allocated to the subsystems, the architecture of which is being assessed. However, it is critical to ensure that such goals and requirements do exist so that the architecture support for enabling the system to achieve these goals and meet these requirements can be assessed. Therefore, QUASAR *does* include a Subsystem Requirements Review Phase during which the quality and maturity of these goals and requirements is reviewed so that any problems can be fixed in time for these quality goals and requirements to properly drive the development of the architecture and for the architecture to be properly assessable.

4. Determine requirements compliance.

Regardless of the contractual formality of the customer requirements, the development organization will still need to derive and specify new, more detailed technical requirements at the system level. Some of these requirements will (or at least should) be architecturally significant, and this includes support for quality requirements, functional requirements, data requirements, interface requirements, and architectural constraints.[7] When developing software-intensive systems, architecturally significant requirements will typically be derived and allocated to individual subsystems, lower level subsubsystems, and so on. A major objective of QUASAR system architecture assessment is to determine compliance of the system and subsystem architectures with their derived and allocated architecturally significant requirements.

5. Determine architecture completeness and maturity.

A software-intensive system must typically be developed using an iterative, incremental, and parallel development cycle.[8] During incremental and iterative development, the ar-

---

[7] Note that the quality requirements are not the only drivers of the system architecture. Major system functions equate to cohesive sets of functional requirements, and unlike modern software, systems still tend to be primarily functionally decomposed. This leads to a natural tension between the functional and quality requirements that the system architects must resolve. The system and subsystem architects must also take other business and programmatic drivers into account and perform engineering tradeoffs between them and the quality requirements. For example, it must be feasible to implement the architecture in terms of program budget, schedule, staffing levels, and staffing experience. Mass-produced systems must also be producible in terms of the production costs, the development organization's manufacturing facilities, and supply chain characteristics.

[8] Development is parallel if multiple activities (e.g., requirements engineering and architecting or the engineering of multiple subsystems) are performed concurrently.

chitecture of a system or subsystem must reach a minimum level of completeness and maturity, respectively, if it is to be properly and completely assessed. Architectural decisions must have been made and properly documented before it becomes possible for the quality of the resulting architecture to be assessed. Typically, an assessment should be postponed if the architecture being assessed is not ready for the assessment.

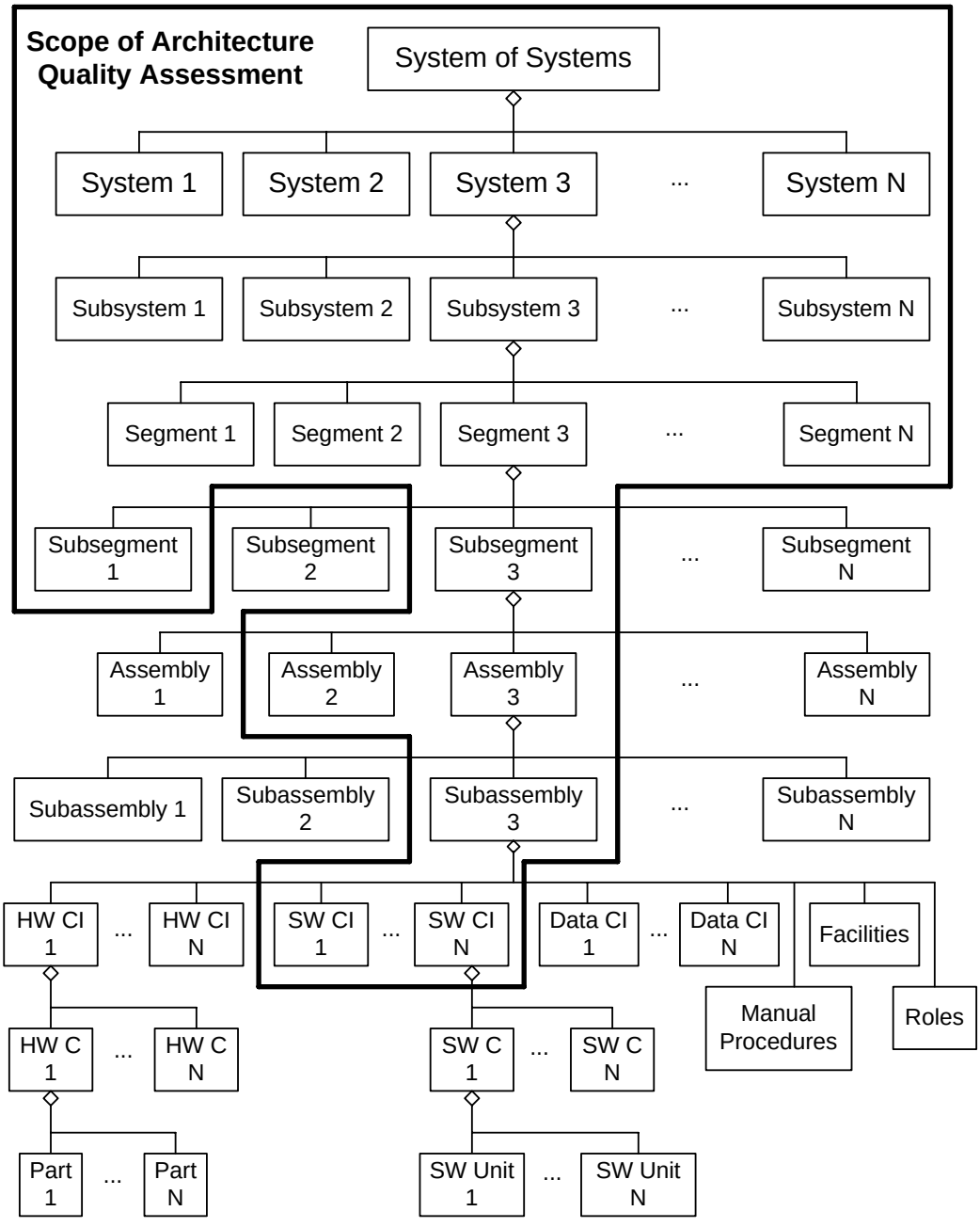

Figure 1: Assessment Scope in Terms of Subsystems[9]

[9] In Figure 1, the acronym CI denotes *configuration item*, while C indicates *component*. Architectural elements below CI should probably be considered to be part of the design rather than part of the architecture.

Any non-trivial system is typically decomposed into subsystems, sub-subsystems, and so forth. As illustrated in Figure 1, this decomposition hierarchy of systems into subsystems is often organized into a series of horizontal tiers (e.g., system of systems, systems, subsystems, and sub-subsystems[10] such as segments, subsegment, assemblies, subassemblies, and so on) to manage complexity and improve human understandability. Architecturally significant requirements will be iteratively and incrementally engineered, with new, ever-more-detailed requirements being derived and allocated to lower level architectural elements. The architectures of these subsystems will be iteratively and incrementally developed in parallel with their allocated requirements. This clearly means that some subsystem architectures will be available for assessment before others, making it impossible to assess all subsystem architectures simultaneously, unless one waits until the entire system architecture is completed. However, waiting to the end of system architecting before assessing the architecture of existing subsystems eliminates the advantages of the early assessment of architecture. Instead, one must incrementally and iteratively assess the quality of these subsystem architectures before they become "chiseled in granite" with too much design and implementation being based on them to make fixing any architectural defects or sub-optimizations practical.

System architecture quality assessments should be performed in a top-down manner, tier by tier, as part of the natural top-down incremental decomposition of the system architecture.[11] Some subsystem assessments are scheduled before others within given builds (blocks or increments[12]), some subsystem assessments are postponed until later builds, and some subsystem assessments will only be partially performed during one build and completed during later builds when the remaining parts of the subsystem architecture are completed. Thus, the growing completeness and maturity of the architecture will have a significant impact on the scope and schedule of architecture assessments.

As illustrated in Figure 1, the scope of the overall system architecture quality assessment is determined iteratively and incrementally, based on the decomposition of the system into lower level tiers of subsystems, sub-subsystems, and so on. This figure shows the

---

[10] Note that the number of tiers and the naming conventions used for the architectural elements at the different levels of the architecture is relatively arbitrary. From this point on, we will, for the sake of simplicity, use the term *subsystem* to mean any lower level architectural element, regardless of its tier (level) in the system decomposition hierarchy.

[11] Whereas performing architecture assessments in an incremental, top-down manner fits well with modern development cycles, it raises the problem of how to assess the overall highest level architecture of an entire system or system of systems. Huge heterogeneous systems are often decomposed into top-level subsystems that have little architecturally in common with each other and are developed by unrelated organizations (e.g., subcontractors). Thus, there may be little in the way of overall architecture integrity, and most of the actual architecture decisions reside at lower tiers of the architecture.

[12] Although the term *increment* is often used to label a single pass through the development cycle, use of this term is misleading. Modern development cycles are incremental, but they are *also* iterative, parallel, and time-boxed.

hypothetical boundary of such a system architecture quality assessment based on the criticality of the quality of the lower level subsystem architectures to the quality of the overall architecture, as well as the available resources that can be invested in performing the lower level assessments. Basing the scope of assessment on the size, complexity, and criticality of the subsystems to be assessed typically results in a ragged lower boundary between those subsystems that are assessed and those that are not.

In addition to influencing the scheduling of the assessments, the completeness and maturity of the architecture is important for measuring project progress, especially when developing software-intensive systems using an iterative and incremental development cycle. Because architecture completeness and maturity is not indicated by the completion of a single architecture document, management and customer oversight require an objective and independent assessment of architecture completeness and maturity. To provide management with independent and objective metrics describing the development status of the system and subsystem architectures, it is often important to independently assess the status of these architectures.

Because a minimal level of architectural maturity[13] is needed before a system architecture quality assessment can be successfully performed, stakeholders can ask the following questions when scheduling assessments:

- **Requirements engineered?**
  Have the relevant architecturally significant quality requirements been properly derived and allocated to the subsystems, the architecture of which is to be assessed?
- **Architecture exists?**
  Does all of the system/subsystem architecture within the scope of the assessment exist? Will all of the architectural decisions have been made by the time that the architecture is to be assessed?
- **Architecture properly documented?**
  Is this architecture adequately and properly documented in terms of official project architectural diagrams, models, and documents? By "official," we mean formally planned project documentation under configuration control, rather than informal, temporary documentation (e.g., slides developed just for the sake of the assessment).
- **Milestones?**
  Will the architecture to be assessed be sufficiently complete so that the assessment results can support a major programmatic milestone such as Preliminary Design Reviews (PDRs) and Critical Design Reviews (CDRs)?

In light of these questions, an important secondary objective of the QUASAR assessment is to perform an objective, independent assessment of the completeness and maturity of the system architecture in terms of the completeness and maturity of its associated subsystem architectures so that the results can be reported to stakeholders.

[13] An architecture can be considered "mature" when its rate of iteration (change) slows to the point where it becomes relatively stable.

6. Determine architecture consistency and integrity.

The development of a software-intensive system involves the successive decomposition of the system into subsystems, sub-subsystems, and so on. Unfortunately, because of schedule constraints and other factors, it is difficult or impossible to develop a system's architecture in a truly top-down manner. Subsystem architectures are developed in parallel with each other as well as in parallel with the architecture of their higher level parent subsystems. Therefore, inconsistencies tend to develop between subsystem architectures, creating a danger that architectural integrity will be lost.

Architectures are consistent to the extent that they incorporate the same architectural decisions to solve similar problems in similar contexts. They tend to incorporate the same architectural styles, patterns, and mechanisms. This leads members of the architecture and assessment teams to ask the following questions concerning architecture consistency:[14]

- **Consistent vertically?**
  Are the architectures of subsystems consistent with the architectures of related subsystems at both higher levels and lower levels in the overall system architecture's decomposition hierarchy?
- **Consistent horizontally?**
  Are the architectures of related (e.g., interoperating) subsystems at the same level of the tier structure consistent?
- **Consistent across specialty architectures?**
  Is there consistency among the various types of specialty architectures such as the hardware architecture, software architecture, database architecture, safety architecture, and security architecture?
- **Consistent support for quality factors?**
  Are the architectural decisions made to support the various types of quality requirements consistent across subsystems? Are the architecture decisions supporting individual types of quality requirements consistent with the types of engineering trade-offs that must be made between competing quality factors (e.g., increasing performance may decrease maintainability and increasing security may decrease usability)?
- **Consistent across development groups?**
  Are architectures consistent across multiple
  - corporations (e.g., prime contractor and subcontractors)?
  - organizations and teams (e.g., subsystem development teams) within a single corporation?
  - groups at distributed development locations?

[14] Two architectures are consistent to the extent that they incorporate the same architectural decisions to solve similar problems in similar contexts. Consistent architectures tend to incorporate the same architectural styles, patterns, and mechanisms.

QUASAR assessments can be used to determine the consistency of the subsystem architectures of an overall system's architecture. As a way of identifying a lack of architectural consistency, QUASAR assessments can help the system architects enforce architectural integrity down through the system architecture's aggregation hierarchy of subsystems.

7. Help architects develop, document, and improve their architectures.

Developing a good quality architecture that meets all of the architecturally significant requirements is a very difficult task, especially given the size and complexity of many of today's systems and especially systems of systems. The architecture of modern systems is huge, consisting of many subsystems, sub-subsystems, and so on, that must collaborate in complex ways to meet huge numbers of stringent requirements. The behaviors of such systems are also highly complex. By ensuring the timely existence of architecturally significant requirements and by helping to ensure that the architect's focus remains on achieving these requirements, the architects are more likely to correctly make and properly document the associated architectural decisions. A major objective of QUASAR assessments is therefore to help the architects create and document their architectures and properly prepare for the associated assessments. The architects can then identify the architectural defects and weaknesses found during the assessments and improve their architectures moving forward.[15]

8. Identify architecture defects early.

The quality of a system's architecture is critical because any defects of either commission (poor architectural decisions) or omission (missing architectural decisions) in the architecture will flow down into lower level subsystem architectures as well as the resulting designs and implementation. Thus, the assessment should begin to verify the architecture's support for architecturally significant quality requirements early, prior to significant design so that any areas of non-compliance can be corrected with an acceptable negative impact on the cost and schedule of the project. In other words, the assessment should identify architecture defects and risks during architecture development as opposed to after the system has been implemented and is under test when architectural defects are difficult and costly to repair. A major objective of QUASAR assessments is thus to identify architectural (strategic) defects early in the development cycle and before they are propagated throughout they system, resulting in many design and implementation (tactical) defects.

9. Manage architectural risks.

Because the architecture of a software-intensive system is large and complex, it will be practically impossible for the system architects to get everything right at the start of the architecture development. Combining the probability of architectural defects with the serious harm that they typically cause yields a significant system architecture risk. Thus, a

[15] Defects and weaknesses are found by the architects when preparing for the assessments and by assessors during the actual assessment. Both lead to iterative improvement of the architecture.

major objective of QUASAR assessments is to help lower these architectural risks to an acceptable level by lowering the probability of architectural defects and increasing the probability that the remaining defects are found sufficiently early in the development cycle so that the can be fixed and their negative consequences either eliminated or mitigated. Major subordinate QUASAR objectives are to

- identify major architectural risks, as well as strengths, to the extent practical given the limited duration and depth of the assessments
- make an initial estimate of the criticality (magnitude of negative impact) and probability of occurrence of the architectural risks that are identified
- identify system architecture risks early in the program when they can be effectively mitigated
- identify quality requirements that are not adequately supported by the architecture so that they can be tracked

10. Provide stakeholder visibility into the architecture.

A system's architecture has many stakeholders that need timely visibility into the architecture. Customer and user organizations want visibility into the architecture to ensure that the system they are acquiring will meet the architecturally significant requirements and possess the necessary qualities. The requirements team wants visibility to ensure that the architecturally significant requirements they engineered will be met by the resulting system. The designers, implementers, and testers want confidence that the architecture that drives their work has sufficient quality and will not need to be significantly altered, thereby invalidating their designs, implementations, and integration tests. Managers want to verify that the architecture can be implemented within budget and schedule constraints. A properly performed architecture assessment provides either direct visibility into the architecture, if the stakeholders are part of the assessment team, or provide them confidence that they can rely on the architectural models and documents, if they are not part of the assessment team.

Under current DoD acquisition practice, the defense contractor is given either capability or performance-based requirements and is responsible for the development and delivery of a system that meets these requirements. Typically, the defense contractor has little obligation to deliver specific architecture documentation to the program management office (PMO) early in the development process. The PMO also needs visibility into the technical architecture of the system; a series of system architecture assessments is one way for engineering personnel in the PMO to obtain that visibility and leverage.

11. Provide acquisition oversight of the architecture.

If the acquiring organization is funding the development of the system, then it assumes acquisition oversight responsibilities. It is important that the acquirer ensures that the system is being developed appropriately. This includes ensuring that the system architecture is being properly developed by assessing and approving development work products such as architectural models and documentation. A system architecture assessment is an appropriate method for doing so.

A system architecture assessment is a technical evaluation of the architecture's quality and as such should not typically be part of major milestone reviews, which are at a higher managerial level. Milestone reviews are often used by the defense contractor to convince the PMO that they have achieved a specific milestone, thereby justifying the associated payment of funds. System architecture quality assessments should instead be performed incrementally and iteratively prior to the associated milestone reviews, which need only summarize the results of the system architecture assessments.

This being said, it is critical that acquisition personnel understand the quality and completeness of the system architecture as well as any significant architectural risks at the major milestone reviews such as PDRs or CDRs. Thus, the system architecture quality assessment becomes the basis for the PMO's approval of the architecture and system at the milestone reviews.

If system architecture quality assessments are to be used to technically prepare for major milestone reviews, then this should be included in the request for proposal (RFP) and contract.

12. **Develop consensus.**

    The assessment should ensure that a consensus between the teams is developed regarding the meaning of the quality factors and the associated quality requirements that the architecture being assessed must fulfill.[16]

13. **Ensure usability of architecture documentation.**

    The assessment should help ensure that the architecture's documentation is usable by stakeholders like acquisition personnel, users, developers, testers, safety and security engineers, and maintainers.

    – Is the system architecture documented in a format that is usable by all the relevant stakeholders?
    – Is the system architecture documentation organized in a way that allows stakeholders to locate information of interest?
    – Does the system architecture documentation provide adequate detail without overwhelming readers?[17]

The objectives of system architecture quality assessments vary depending on the organization that is responsible for performing the assessment. Assessments, for example, may be performed by the

- **architecture team**, as part of a team-internal technical assessment
- **development organization**, as part of an organizational-internal quality assessment
- **customer organization**, as part of its independent oversight or verification duties

[16] If consensus cannot be reached, an action item must be created and tracked to resolution.

[17] Although this is difficult to achieve in practice, it nevertheless should be a goal to be strived for.

2.2 Philosophy of Architecture Assessment

- **Quality requirements drive the architecture.**

  An architecture is insufficient if it only supports the performance of its allocated functional requirements. Rather, the quality of the architecture (and acceptability of its systems) is largely based on how well the architecture also supports its allocated quality requirements. This is true whether or not these quality requirements have been explicitly derived and allocated to the architecture. Thus, it is not just *what* the architecture enables a system to do, but rather *how well* it enables the system to do it (i.e., how well it supports its allocated "-ilities"). For example

  – The architecture may be layered and modular to support the meeting of maintainability requirements.
  – The architecture may include a commercial off-the-shelf (COTS) real-time operating system and deterministic scheduling to support the meeting of performance requirements.
  – The architecture may include redundant hardware to support the meeting of availability and reliability requirements.

- **The safety cases approach can be generalized for quality cases.**

  The safety case approach developed within the safety community can be generalized and reused for other quality factors. Specifically, it forms the basis for an architecture assessment method for verifying the sufficiency of architectural support for other types of quality requirements.

- **Architects make their cases to assessors.**

  The architects are responsible for successfully making their case to the assessors that their architecture sufficiently supports achieving the architecturally significant quality requirements. The assessment team should not have to work hard to determine what the architects have done. The assessment team's liaison to the architecture team should not have to make the architecture team's arguments and identify and provide their evidence for them.

- **Arguments must be clear and compelling.**

  To pass the assessments, the architects must convince the assessors that the architecture adequately supports the system's meeting its quality requirements. Therefore, the architects' arguments (i.e., presentation and documentation of architectural decisions and associated rationales) must be clear and compelling.

- **Evidence must be credible.**

  Unless the architects provide proper evidence, their case that the architecture adequately supports its derived and allocated quality requirements is not credible. Relying merely on the verbal assurances of the architects boils down to little more than the "trust me" argument. The cases must be based on real evidence; the evidence is not real unless it is in the project's official architecture documentation (e.g., under configuration control).

Verbal discussions and "quick and dirty" PowerPoint presentations developed specifically for the assessment are not acceptable evidence, although they can be quite useful in helping to determine how well the architecture team understands its architecture.

- **Assessors probe architects' cases.**

  Although the architects are responsible for successfully making their quality cases to the assessors, it is *not* appropriate to rely on them to make perfect quality cases. After all, it is not uncommon for the architects to be overly confident that their architectural decisions are adequate to ensure that their system and subsystem architectures sufficiently support the ability of the system and subsystems to meet the derived and allocated quality requirements. Architects also naturally tend to present their best arguments and downplay any known weaknesses in their architectures.

  Therefore, assessors should *not* be a passive audience during the architects' presentation of quality cases. For one thing, the assessors are responsible for properly preparing to hear the architects' quality cases; they must learn about the architectures by reading the architects' preparatory materials. More importantly, the assessors are responsible for using this understanding to actively probe the architects' quality cases for any potential weaknesses and risks. Note that to be able to do these two things, the assessment team must contain assessors who are skilled system architects with backgrounds in either relevant specialty engineering areas (e.g., reliability engineering, safety, security) or relevant application domains (e.g., communications, power supply, propulsion, sensors).

2.3 Assumptions

The QUASAR method is based on the following assumptions about the system that is being architected:

1. **The system is large and incrementally and iteratively developed.**

   Assumptions: The system is very large and complex, requiring it to be iteratively and incrementally developed.

   Consequences*:*

   a. The system has a large number of architecturally significant derived requirements that have been allocated to multiple levels within the system architecture.
   b. The system's architecture is very large and complex.
   c. Both the architecturally significant requirements[18] and the associated architecture are being developed in an incremental and iterative manner.

[18] Even if the requirements are "officially" being engineered using a waterfall approach, in practice they will actually be engineered in an incremental and iterative manner. Regardless of whether or not the requirements are being developed incrementally and iteratively, a software-intensive system will almost certainly be architected in this manner. Typically, as the system is decomposed into subsystems and sub-subsystems, these subsystems will be architected in a top-down manner.

d. It is impractical to adequately assess the entire system architecture all in one step; architecture assessments must also be performed in an incremental and iterative manner.

2. Quality requirements are important architecture drivers.

Observations: In practice, system requirements engineering tends to concentrate on functional requirements. Unlike software which currently tends to be decomposed using object-oriented design methods, systems tend to be functionally decomposed into subsystems and sub-subsystems that are functionally cohesive. Quality requirements (e.g., availability, capacity, efficiency, interoperability, modifiability, performance, portability, producibility, reliability, reusability, robustness, safety, scalability, security, testability, and usability) tend to be poorly and incompletely engineered. Quality requirements greatly influence the quality, acceptability, and architecture of the system.

Consequences*:*

a. Because functional requirements and the functional decomposition of systems into functionally cohesive subsystems are emphasized, system architectures tend to be driven by and map reasonably well to the meeting of functional requirements.
b. In practice, the risk that systems will not meet their quality requirements is larger than the risk that they will not meet their functional requirements.
c. Because quality requirements are primary drivers of the architecture, the architecture assessment method should concentrate on assessing whether or not the architecture sufficiently helps the system and its subsystems meet their derived and allocated quality requirements.

3. Quality requirements are often missing or of poor quality.

Observations: In practice, many architecturally significant quality needs are never specified as quality requirements and many of those that are specified are poorly specified. They tend to be vague goals such as “the system shall have high availability” or “the subsystem shall highly interoperable” rather than actual requirements that are complete, consistent, feasible, unambiguous, and verifiable.

Consequences*:*

a. The maturity and quality of the quality requirements must be reviewed sufficiently early to enable them to be corrected in time for them to drive the architecture and to enable the architecture to be properly assessed.
b. Quality goals and requirements are often pushed down “as is” to lower level subsystems without being properly derived to be more detailed and specific.

3 Quality Cases

Each subsystem architecture team is responsible for convincing the assessment team that their subsystem architecture adequately supports the subsystem's ability to fulfill the derived quality goals and meets the derived quality requirements that have been allocated to the subsystem. After all, who else better knows the

- quality goals and quality requirements that drove their architectural decisions
- architectural decisions (e.g., selection of architectural patterns and use of architectural mechanisms) they made and why they made them
- where in the architectural documentation they explicitly document decisions and associated rationales

In other words, the members of the subsystem architecture team

- make *claims* to the assessment team that the subsystem architecture adequately supports the subsystem's ability to
  - fulfill the derived quality goals
  - meet the derived quality requirements
- present clear and compelling *arguments* to the assessment team as to why the assessment team should believe that
  - they made certain architectural decisions
  - they made appropriate decisions
  - the combination of these decisions is adequate to justify belief in their claims
- provide sufficient *evidence* to the assessment team supporting their arguments
  In other words, trustworthy evidence that
  - they actually made the architectural decisions that they argue they did
  - these were appropriate and sufficient decisions to justify belief in the subsystem architecture team's claims

In the QUASAR method, the way for a subsystem architecture team to make their case to the assessment team is to produce a set of *quality cases* and formally present them to the assessment team. Thus, the foundation on which the QUASAR method is based is the concept of a quality case, which is a generalization of the safety case approach developed within the safety community.

3.1 Definition of Quality Cases

As illustrated in Figure 2, *quality factors* (e.g., interoperability, performance, safety, or security) define a single type of quality of a system or subsystem. A quality factor is typically decomposed into one or more *quality subfactors*. For example, the quality subfactors of performance are jitter, latency, response time, schedulability, and throughput. Because a single quality case is specific to a quality factor or one of its quality subfactors, quality cases can be classified as interoperability cases, performance cases, safety cases, security cases, and so on.[19] A quality case consists of a set of related

- **Claims**

  Claims are the developers' assertions that the system or subsystem adequately achieves its allocated quality goals and meets its allocated requirements.

- **Arguments**

  Arguments are the developers' clear and compelling reasons that justify belief in the associated claims (i.e., reasons why the assessors should believe that the system or subsystem adequately meets its allocated goals and requirements).

- **Evidence**

  Evidence is sufficient credible documentation (or witnessed demonstrations) that supports the developers' arguments.

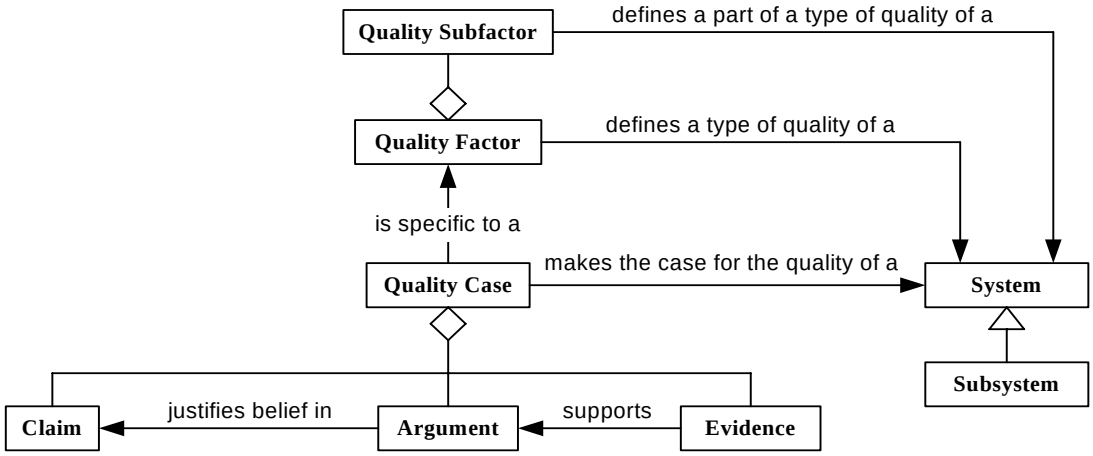

Figure 2: Structure of Quality Cases

Typically, a quality case is constructed near the end of development and its arguments and evidence address all major project disciplines (requirements, architecture, design, implementation, integration, and testing). This allows the quality case to make a strong case for the

[19] Note that this is the structure of a general quality case and is not restricted to an architecture-level quality case.

quality of the *completed system* or one of its *completed subsystems.* However, such a complete quality case would be constructed and presented much too late in the development cycle to be used as the foundation for an architecture quality assessment.

Therefore, the QUASAR method uses *architecture quality cases* that are restricted to architectural information. As illustrated in Figure 3, an architecture quality case consists of

- **Architectural Claims**

  Architectural claims are the architects' assertions that the system/subsystem architecture sufficiently supports the system/subsystem ability to achieve its allocated quality goals and meet its allocated requirements.

- **Architectural Arguments**

  Architectural arguments are the subsystem architects' clear and compelling reasons that justify belief in their claims. These are typically the architects' architectural decisions (e.g., use of appropriate architectural components, mechanisms, or patterns) that compose the reasons why the assessors should believe that the architecture adequately supports the fulfillment of the derived quality requirements that have been allocated to the subsystem.

- **Architectural Evidence**

  Architectural evidence is sufficient credible[20] evidence to support the architects' arguments (e.g., adequate official project architecture diagrams, architecture models, architecture documents, and executable architectural prototypes).

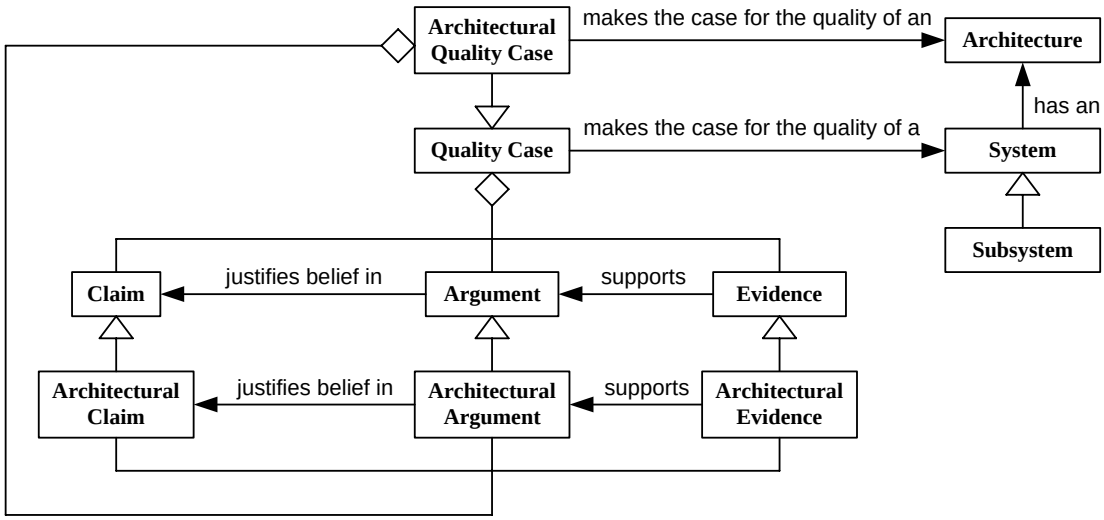

Figure 3: Structure of Architectural Quality Cases

[20] To be credible, evidence should be official, relevant, correct, current, and under configuration control.

3.1.1 Claims

In a quality case, a *claim* is defined as an assertion made by the development team to the assessment team that the system or subsystem either adequately achieves one or more related quality goals or meets a set of one or more associated quality-related requirements that have been allocated to it.

The following are important consequences of this definition:

- A claim is not just any *assertion*. A claim is related to one or more related *quality goals* or to an associated set of one or more *quality-related requirements*. A claim is therefore related to a specific quality factor. One claim may be about interoperability, a second claim may be about performance, and a third claim may be about safety. If a relevant quality factor has important quality subfactors, then there may also be claims about them. For example, a performance claim may be decomposed into claims about response time and throughput. The quality factor related claim "subsystem X will have high performance" can be decomposed into the quality subfactor claims "subsystem X will exhibit a rapid response time" and "subsystem X will have a high throughput." As a natural part of requirements engineering, such high-level goals should be engineered into derived performance requirements such as "under normal operating conditions, subsystem X shall respond to stimulus Y within Z milliseconds" and "when in disabled mode A, subsystem B shall on average complete at least C transactions of type D per second."
- During a quality assessment, a development team makes a series of claims to an assessment team with the goal of convincing the assessment team of the validity their claims concerning the quality of the system or subsystem. The development team therefore attempts to justify the assessors' belief in these claims by using clear and compelling arguments based on supporting evidence.
- A claim can be made either about the entire *system* or about one of its *subsystems*. However, more claims are naturally made about subsystems than the overall system because there are more subsystems.
- A claim can be made about either achieving a quality *goal* or meeting a set of associated quality-related *requirements*. Ensuring that the system or subsystem meets a cohesive set of consistent, feasible, unambiguous, and verifiable requirements is almost always much more important than the achievement of more general goals.

Architectural quality cases contain system or subsystem architectural claims rather than the more general system or subsystem claims. These architectural claims are either about the architecture of a system or about the architecture of one of its subsystems. In an architectural quality case, an *architectural claim* is defined as an assertion made by an architecture team to an architecture assessment team that the system or subsystem's architecture adequately helps the system or subsystem either achieve one or more related quality goals or meet a set of one or more associated quality requirements that have been allocated to it.

The preceding definition of an architectural claim implies that

- The architectural claim is limited. Just because a system or subsystem has an adequate architecture does not mean that the system or subsystem will achieve its quality goals or will meet all of its associated quality requirements after it is implemented. Defects in design, implementation, and integration cannot be overcome by the architecture, no matter high its quality. Having a good architecture is a necessary but insufficient condition for achieving overall quality goals and meeting requirements.
- An architectural claim can be made either about the system-level architecture or about the architecture of one of its subsystems. As before, more claims are made about subsystem architectures than the overall system architecture because there are more subsystems.
- An architectural claim can be made about either achieving a quality goal or meeting a set of related quality requirements.

As illustrated in Figure 4, there are many different kinds of architectural claims that the system or subsystem architecture teams could make. First of all, there are many different kinds of quality factors and quality subfactors that might be important drivers of the architecture. Secondly, claims can be about achieving quality goals or meeting quality-related requirements. The claims about achieving quality goals can be further subdivided into goals related to the overall quality factor or to one of its quality subfactors. Quality-related requirements can also be about meeting minimum mandatory levels of quality factors and subfactors, but they can also be about other related requirements. Unlike quality requirements, quality-significant requirements are any normal functional, data, or interface requirements with quality ramifications. Quality subsystem requirements are requirements for any subsystem, the sole purpose of which is to achieve that quality. Thus, *all* of the requirements for a fire detection and suppression subsystem are quality subsystem requirements. Finally, constraints mandating certain quality mechanisms are called *quality constraints*.

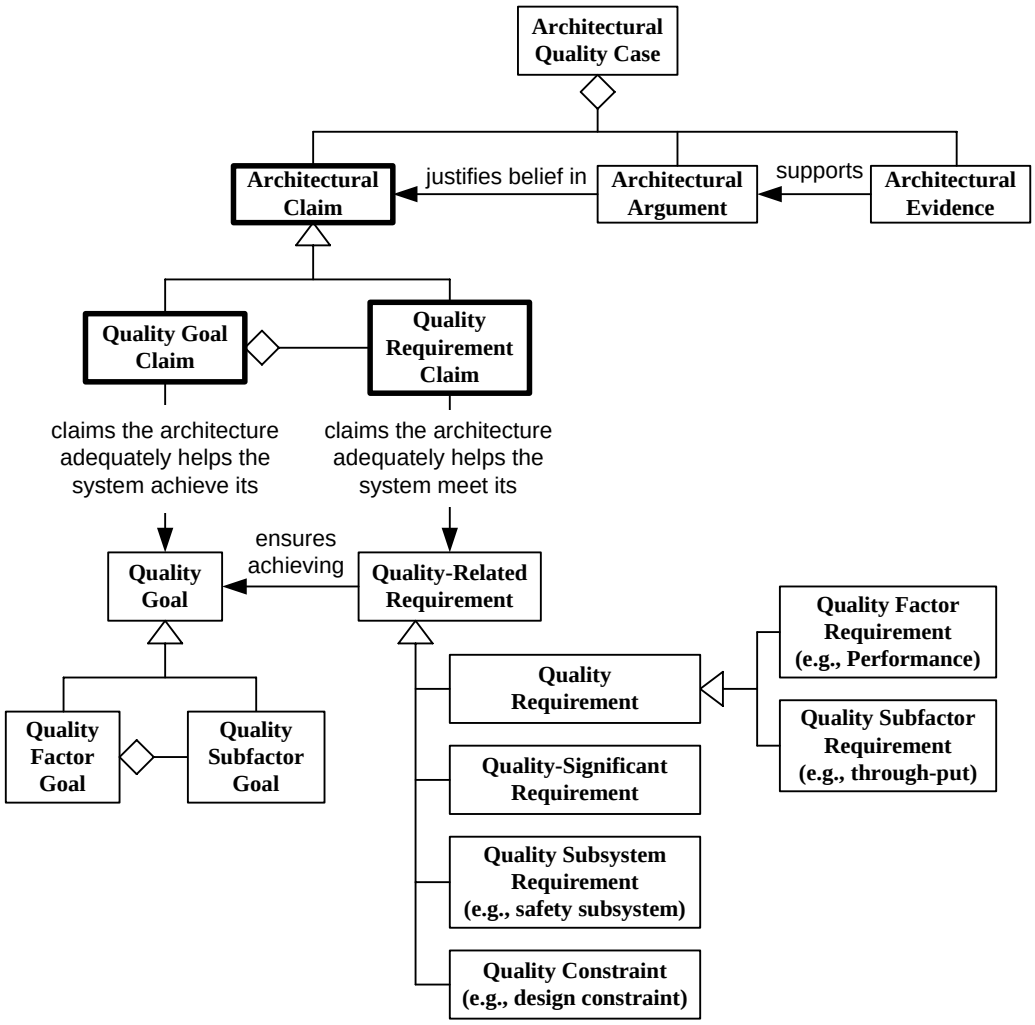

Figure 4: Types of Architectural Claims

Consider an automated people mover (APM) system decomposed into a taxi subsystem (i.e., the taxis), a guideway[21] subsystem (i.e., the concrete guideways on which the taxis drive), a taxi station subsystem, and a central control subsystem. The following are examples of the different kinds of safety claims that the architects might make when constructing its safety cases:

[21] *Guideway* is the industry standard term for the path (e.g., concrete road, monorail, or train tracks) along which an APM travels.

1. **Safety Goal Claims**
   a. **Safety Factor Goals**
      – *System Is Safe to Use and Operate*
        The architecture of the automated taxi system adequately supports the system achieving the following safety factor goal: “The automated taxi system will be safe to use and operate.”
      – *Taxis Are Safe to Use*
        The architecture of the automated taxi subsystem adequately supports the subsystem achieving the following safety factor goal: “The automated taxi subsystem will be safe for its passengers to use.”
   b. **Safety Subfactor Goals**
      – *System Prevents All Accidents*
        The architecture of the automated taxi subsystem adequately supports its achieving the following harm prevention safety subfactor goal: “The automated taxi subsystem will help it to avoid injuring its passengers.”
      – *System Detects All Guideway Hazards*
        The architecture of the automated taxi subsystem adequately supports its achieving the following hazard detection safety subfactor goal: “The automated taxi subsystem shall detect all guideway hazards (e.g., stalled vehicle or unlocked switch).”
      – *System Reacts to All Accidents*
        The architecture of the automated taxi subsystem adequately supports its achieving the following accident reaction safety subfactor goal: “The automated taxi subsystem shall properly react to all detected accidents.”
2. **Safety-Related Requirements Claims**
   a. **Safety Requirements**
      – *Taxi Shall Avoid Rear-Ending Other Taxis*
        The architecture of the automated taxi subsystem adequately supports its meeting the following safety requirement: “A taxi shall not rear-end a taxi in front of it more than twice a year, whereby rear-ending means a collision with a relative impact speed of 1 kilometer per hour or more.”
      – *Taxi Shall Detect Collisions*
        The architecture of the automated taxi subsystem adequately supports its meeting the following safety requirement: “At least 99.9% of the time, a taxi shall detect if it collides with another taxi with a relative speed of more than 2 kilometers per hour.”
      – *Taxi Shall Report Collisions*
        The architecture of the automated taxi subsystem adequately supports its meeting the following safety requirement: “At least 99.9% of the time upon detecting a collision, the automated taxi subsystem shall notify the command subsystem within 5 seconds that it has been involved in the collision.”
   b. **Safety-Significant Requirements**
      – *Taxis Shall Know Velocity*
        The architecture of the automated taxi sensor sub-subsystem (SS) adequately supports its meeting the following safety-significant requirement: “The automated taxi SS shall determine the velocity of the taxi relative to the guideway

with an accuracy of 0.2 kilometers per hour and a precision of within ± 0.1 meters per second."

– *Taxis Detect Headway*
The architecture of the automated taxi SS adequately supports its meeting the following safety-significant requirement: "The automated taxi SS shall detect if the next taxi in front of it gets within its headway (i.e., the safe stopping distance) [as indicated by table X]."

– *Taxis Obey Speed Limit*
The architecture of the automated taxi power braking subsystem (PBS) adequately supports its meeting the following safety-significant requirement: "Subject to guideway speed limits, the PBS shall be able to accelerate the taxi up to a maximum forward velocity of 80 kilometers per hour."

c. Safety Subsystem Requirements

– *Smoke Detector Sensitivity*
The architecture of the automated taxi fire detection and suppression subsubsystem (FDSS) adequately supports its meeting the following safety subsystem requirement: "At least 99.9% of the time, the FDSS shall detect smoke particles larger than 0.01 micrometers in concentrations more than 10,000 particles per cubic centimeter."

– *Smoke Detection Reliability*
The architecture of the FDSS adequately supports its meeting the following safety subsystem requirement: "The FDSS shall have a reliability of more than 99.9%."

d. Safety Constraints

– *Seat Belts*
The architecture of the automated taxi subsystem incorporates the following safety constraint: "The automated taxi subsystem shall provide standard COTS-based automotive seat belts for all passengers.

– *Safety Glass*
The architecture of the automated taxi subsystem incorporates the following safety constraint: "The automated taxi subsystem shall use COTS safety glass for all windows."

3.1.2 Arguments

In a quality case, an *argument* is defined as a reason given by a development team to an assessment team that justifies belief in a claim. Therefore in an architecture quality case, an *architectural argument* is defined a reason given by an architecture team to an architecture assessment team that justifies belief in an architectural claim. The following are important consequences of this definition:

- Arguments should be clear and compelling if they are going to convince the assessment team that belief in the claims is justified.
- Arguments should be backed up by sufficient, legitimate evidence.

- As illustrated in Figure 5, arguments are a combination of the architects'
  - architectural decisions such as the
    - use of appropriate architectural pattern, style, or mechanism
    - incorporation of a specific architectural component
    - use of a specific general way architectural components should collaborate to meet the allocated quality goals and quality requirements
  - rationales for why the architectural decisions are appropriate and adequate

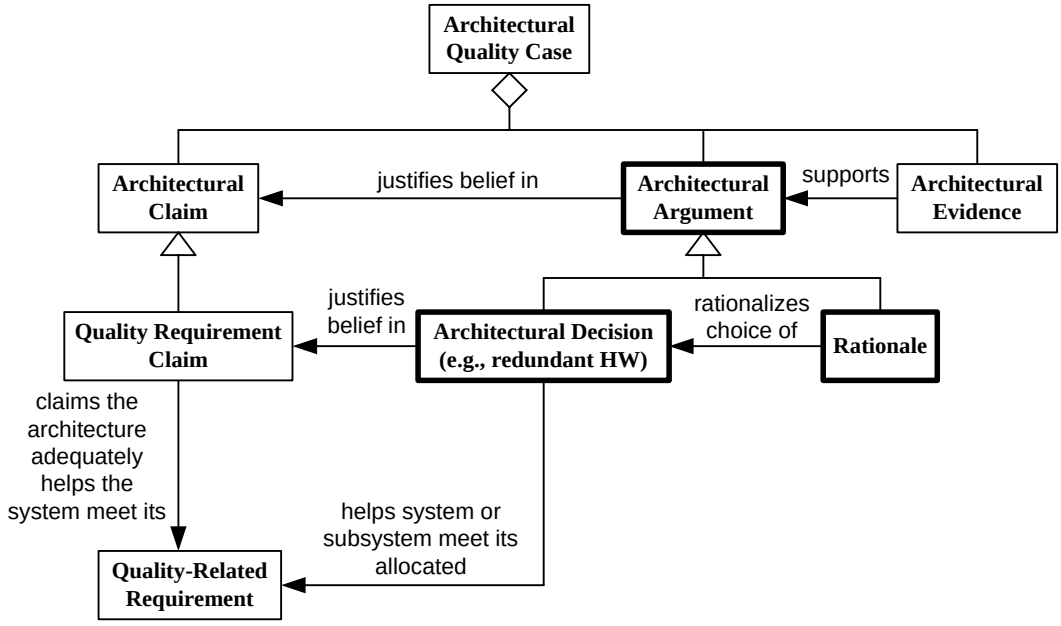

Figure 5: Structure of Architectural Arguments

The following are examples of the different safety arguments that architects might make when building safety cases for the architecture of an APM:

1. Arguments for Meeting Safety Requirements

a. Architecture Decision: Redundant Headway Sensors

The automated taxi subsystem (ATS) incorporates redundant ultrasound and laser sensors for determining if the automated taxi's headway is less than its current safe stopping distance.[22]

Rationale:

These sensors can be used to ensure that the taxi maintains an adequate headway so that it can apply brakes to avoid colliding with the taxi in front of it.

[22] *Headway* is the distance between an APM vehicle and the vehicle in front of it on the same guideway. Headway is used to determine the minimum safe braking distance to be kept between adjacent vehicles. A guideway is the concrete road, railroad tracks, or monorail on which vehicles travel.

b. Architecture Decision: Radio Transmitter

The ATS incorporates a COTS very high frequency (VHF) radio transmitter and receiver. Specifically, it incorporates model X radio from vendor Y.

Rationale:

The radio will be used to notify the central control subsystem of accidents and hazards. The selection was based on an industry trade-study, an analysis of vendor-supplied technical documentation, and a test of prototypes across the range of the taxis.

2. Arguments for Meeting Safety-Significant Requirements

a. Functional Requirement: Close Taxi Door

– *Architecture Decision: Door Resistance Sensor*
The door subsystem (DS) of the ATS incorporates a door resistance sensor on the door motor.
Rationale:
The door resistance sensor enables the automated taxi subsystem to determine if it its doors are closing on a passenger if a maximum safe torque value is exceeded. This enables the door subsystem to reopen the doors and automated taxi subsystem to know not to move when the door is jammed open.

– *Architecture Decision: Maximum Door Motor Torque*
The DS of the ATS incorporates a COTS door motor that has a safe maximum torque rating.
Rationale:
The door motor is not strong enough to seriously crush a passenger.

– *Architecture Decision: Door Lock*
The DS of the ATS incorporates a door lock.
Rationale:
The door lock can be used to ensure that the door remains closed while the taxi is moving.

– *Architecture Decision: Door Lock Sensor*
The DS of the ATS incorporates a door lock sensor.
Rationale:
The door lock sensor can be used to ensure that the door remains locked while the taxi is moving.

– *Architecture Decision: Speed Sensor*
The power braking subsystem (PBS) of the ATS incorporates a speed sensor.
Rationale:
The speed sensor enables the ATS to ensure that the taxi does not move until the door is closed and locked.

– *Architecture Decision: Maximum Door Closure Distance*
The DS of the ATS incorporates a door stop.
Rationale:
The door stop ensures that a 1.5 inch gap remains when the door is closed and locked, thereby ensuring that the door does not crush passengers' fingers.

3. Arguments for Meeting Safety Subsystem Requirements

a. Safety Subsystem: Fire Detection and Control Subsystem

– *Architecture Decision: Smoke Detector*
The fire detection and control subsystem (FDCS) incorporates a COTS smoke detector in the PBS.
Rationale:
An overheated electric motor or brake can generate smoke before catching fire.

– *Architecture Decision: Heat Sensor*
The FDCS incorporates a COTS heat sensor in the PBS.
Rationale:
The high temperature of an overheated electric motor or brake can be detected before they catch fire.

– *Architecture Decision: Automated Fire Extinguisher*
The FDCS incorporates a COTS fire extinguisher in the PBS.
Rationale:
The high temperature of an overheated electric motor or brake can be detected before they catch fire.

– *Architecture Decision: Taxi Speed Override*
The FDCS stops the automated taxi door in case of fire.
Rationale:
Stopping the taxi enables passengers to exit the taxi onto the guideway footpath.

– *Architecture Decision: Door Lock Override*
The FDCS unlocks the automated taxi door lock in case of fire.
Rationale:
Unlocking the taxi door enables passengers to exit the taxi onto the guideway footpath.

4. Arguments for Meeting Safety Constraints

a. Architecture Decision: Seat Belts

The ATS provides standard COTS automotive seat belts for all passengers.

Rationale:

This decision fulfills the seat belts architecture constraint. It can also protect passengers from injury during collision. Using standard COTS automotive seat belts minimizes price while maintaining quality.

b. Architecture Decision: Safety Glass

The ATS incorporates standard COTS safety glass for all windows.

Rationale:

This decision fulfills the safety glass architecture constraint. It can also protect passengers from injury during collision. Using standard COTS safety glass minimizes price while maintaining quality.

3.1.3 Evidence

In a quality case, *evidence* is official factual information that clearly proves the truth of the architects' arguments. It is what supports their claims that a system or subsystem achieves one or more of its quality goals and meets one or more of its quality requirements.

In an architecture quality case, *evidence* is official factual information that clearly proves the truth of the architects' arguments. It is what supports their claims that a system or subsystem

architecture helps it to achieve one or more of its quality goals or meet one or more of its quality requirements.

Figure 6 shows the two types of architectural evidence and how they relate to the other components of an architecture quality case.

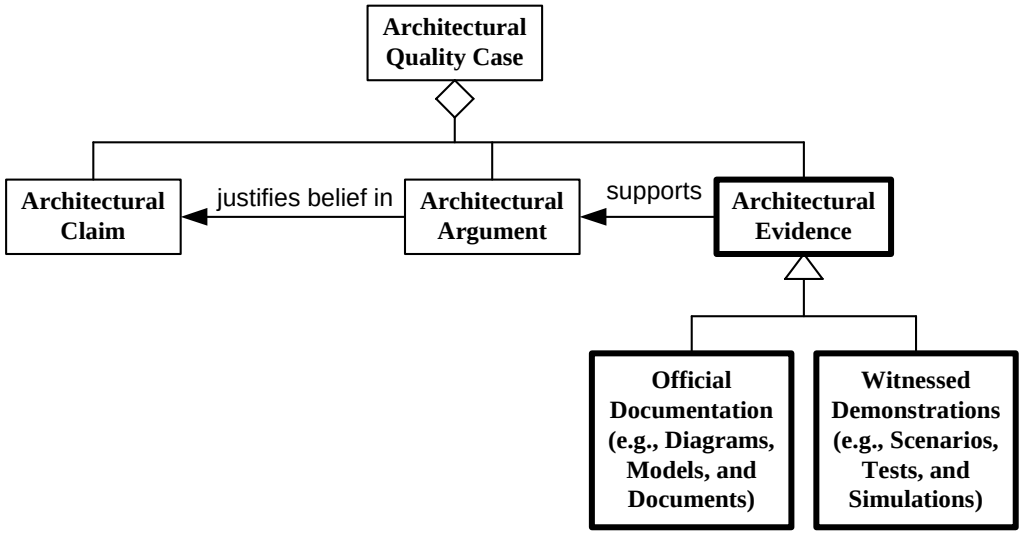

Figure 6: Types of Architectural Evidence

Typical examples of valid evidence that the architects can use to support their arguments include the following:

- **Official Documentation**

  Evidentiary documentation typically includes the *relevant parts* of the following types of official[23] endeavor[24] documentation:

  - **Architecture Documents**
    - *Documents*
      For example, this documentation includes system or subsystem architecture documents or system or subsystem design documents/descriptions.
    - *Presentation Materials*
      For example, this documentation includes system or subsystem architecture presentation materials that are made created for official programmatic milestone reviews, such as PDRs.

[23] Documentation is typically considered to be "official" and appropriate for use as quality case evidence if it is under configuration control, is being maintained, and is intended to be used as a driver for design, implementation, and [especially integration] testing.

[24] The term *endeavor* is used to clarify that the evidence need not be restricted to an individual project. The evidence may belong to a program of related projects (e.g., a product line of programs) or to an entire enterprise of related projects and programs.

- *Training Materials*
  For example, this documentation includes system or subsystem architecture training materials used to train new architects, designers, and testers in the existing architecture.
- *Rule Inspection Results*
  This documentation reports the results of an inspection to determine how well a single architecture follows a common set of architectural rules. These inspection results help determine the quality of an individual subsystem and can also be used to determine architectural integrity across subsystems.
- *Executable Architecture Test Results*
  This documentation includes test reports resulting from the testing of executable subsystem architectures.
- *Functional Description Documents*
  This documentation includes technical documentation of the allocation of logical system functions and their allocation to architectural elements.
- *Interface Description Documents*
  This documentation includes technical documentation describing the data and control interfaces between architectural elements including source, destination, data type, communication medium (e.g., electric, fiber-optic, or radio), and protocol used.
- *Product Technical Documentation*
  This documentation includes vendor-supplied technical documentation describing COTS products used as architectural elements.
- *Product Trade Studies*

  This documentation includes internally developed or commissioned trade studies documenting vendors and their COTS products that have been selected to be used as architectural elements.
- *Quality Factor White Papers*
  This documentation includes system or subsystem interoperability, performance, reliability, safety, and security white papers.
- *Requirements Traceability Matrices*
  This documentation includes requirements traceability matrices that map relevant quality factors or quality-related requirements to the architectural elements that help achieve them.
- *Technology Evaluations*
  This documentation includes trade studies and forecasts of the major technologies to be used including fore cases of their maturity during various states of development and production.

- **Architecture Models**
  - *Data Models*
    These models include logical or physical data models showing data storage or data flows.
  - *Functional Models*
    These models include logical models of the architecture in terms of major system or subsystem functions and the data/control flows between them.

- *Object Models*
  These models include models of the major architectural elements in terms of classes and the relationships (e.g., inheritance, association, aggregation, message passing) between them.
- *State Models*
  These models include models of an architectural element's states and the transitions between these states including substates and transition triggers and guards.

- **Architecture Diagrams**
  - *Activity/Collaboration Diagrams*
    These diagrams show interactions (e.g., messages, data flows, and event traces) between architectural elements and external actors and systems.
  - *Allocation Diagrams*
    These diagrams show the allocation of data and software architectural elements to hardware architectural elements.
  - *Aggregation Diagrams (Configuration Diagrams)*
    These diagrams show how one or more architectural elements are decomposed into their component architectural elements and the aggregation relationships between them (i.e., the aggregation hierarchy).
  - *Context Diagrams*
    These diagrams show the relationships between a blackbox system/subsystem and the external actors and/or systems with which it interacts.
  - *Data Flow Diagrams*
    These diagrams show the major data types that flow between architectural elements.
  - *[Hardware] Block Diagrams*
    These diagrams show the main hardware architectural elements and how they are connected. These are very similar to network diagrams, which concentrate on the networks connecting the hardware components as well as the network connectivity devices.
  - *Hardware Schematics*
    These diagrams show the internal "wiring" of the architectural elements and the physical "wiring" interfaces of the architectural elements.
  - *Layer Diagrams*
    These diagrams show the various logical horizontal levels making up the logical architecture.
  - *Network Diagrams*
    These diagrams show the networks and network connectivity diagrams that enable the hardware architectural elements and external systems to communicate. These diagrams show network types and configurations as well as protocols used.
  - *Statecharts or State Transition Diagrams*
    These diagrams document an architectural element's states and the transitions between these states including substates and transition triggers and guards.
  - *Timing Diagrams*
    These diagrams show how the real-time operating systems perform time slicing and schedule processes.
  - *Wiring Diagrams*
    These diagrams show how architectural hardware elements are wired together.

- **Assessor-Witnessed Demonstrations**
  - *Architecture Simulations*
    Observation of the output of architecture simulations
  - *Executable Architecture Tests*
    Observation of the output of a tests of executable architectures (e.g., architecture prototypes)
  - *Hardware Components*
    Observation of hardware/network physical architecture components and their interconnections in a development laboratory

Although evidence typically consists of current project architectural diagrams, models, and textual documentation that are under configuration control, evidence may also include witnessed demonstrations. For example, demonstrations could include hardware exhibited to and directly observed by the assessment team such as the configuration of the subsystems of a system prototype seen during a tour of a development lab.

Architects often submit documentation that provides evidence of architectural intent, rather than evidence of the actual architecture. Although of some limited value, the following documentation should **not** be considered to be proper architecture quality case evidence because it does not document actual architectural decisions (architecture style, architecture patterns, or architecture mechanisms) made to ensure system architecture quality:

- Architecture Plans
- Architecture Policies
- Architecture Rule Lists (and unfilled-out, associated checklists)
- Architecture Schedules
- Architecture Team Charters and Memberships
- Architecture Standards and Procedures

3.2 Quality Case Diagram

Based on the preceding definitions, Figure 7 illustrates how architecture quality cases can be structured in a fairly standardized manner as a result of their standard content and because they are organized by quality factor and quality subfactors into the following:

- **Architecture Claims** that the architecture adequately helps the system
  - achieve one or more quality goals
    - quality factor (e.g., interoperability, performance, reliability, or security)
    - quality subfactors of the quality factor (e.g., performance can be decomposed into jitter, latency, response time, schedulability, and throughput)
  - meet one or more quality-related requirements
    - quality factor requirement mandating a minimum amount of some quality factor or quality subfactor

– quality constraint mandating the use of some architecture, design, or implementation decision related to the quality factor

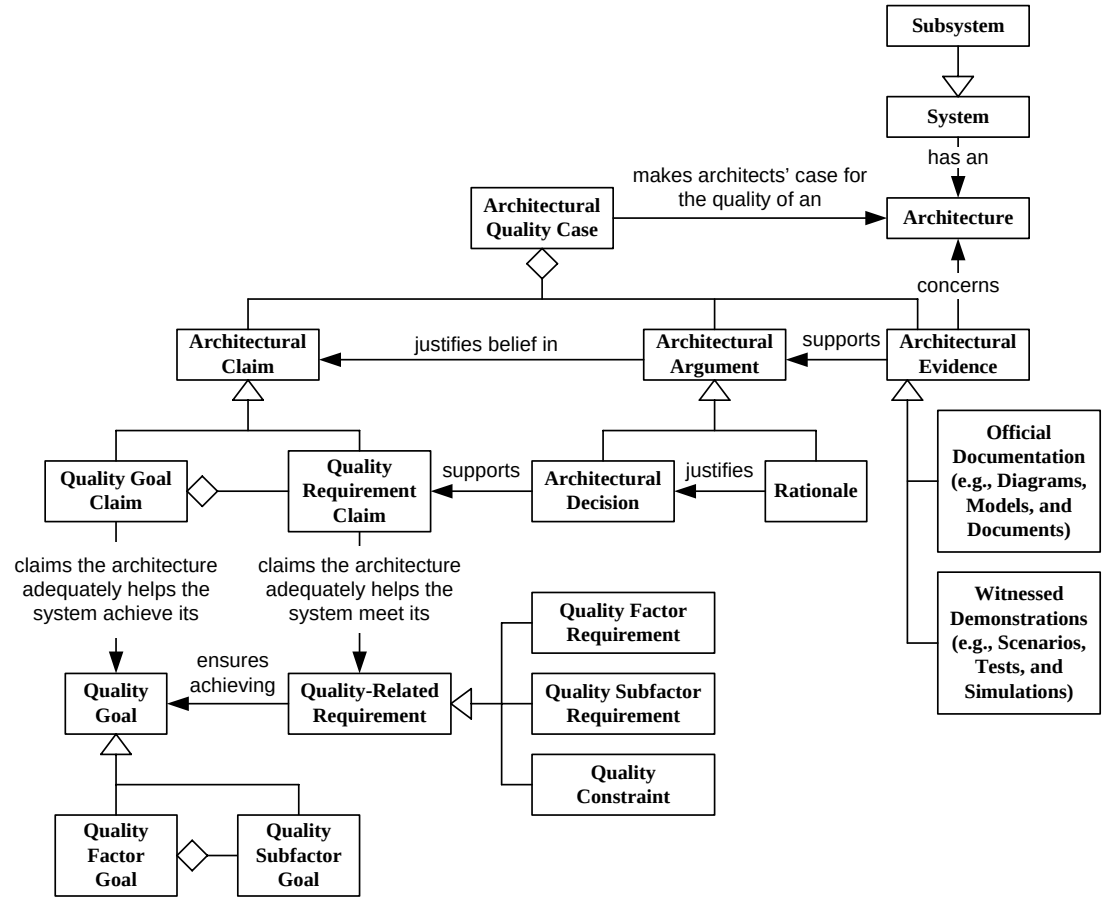

Figure 7: Components of Architectural Quality Cases

- **Architecture Arguments**
  - architectural decisions (e.g., use of architectural mechanisms, patterns)
  - rationale that these architectural decisions give the architecture properties that justify belief in these architecture claims
- **Architecture Evidence** supporting the arguments consisting of *the relevant parts of*
  - official documentation such as
    - documents
    - diagrams
    - models
  - witnessed demonstrations
    - scenarios
    - simulations (e.g., of executable architecture models or specifications)
    - tests (e.g., of executable architectures)

Appendix C.1 defines the most commonly used quality factors and quality subfactors, while Appendix E contains several example architectural quality cases.

Because a software-intensive system may have a large number of significant subsystems needing assessment, a large number of system architecture quality assessments may need to be performed. In addition, a large number of quality factors (see Appendix C) may be sufficiently important to assess each of these significant subsystems against. Thus, each of these subsystem assessments may generate (or make use of) a large number of

1. **Claims** (quality factor and subfactor goals and requirements)
2. **Arguments** (architectural decisions and associated rationales)
3. **Evidence** (relevant parts of diagrams, models, documents, and witnessed demonstrations)

Architects, assessors, and other stakeholders need ways to deal with this complexity. One such way is to organize the quality cases and their component information hierarchically by:

1. **Tier**

   Initially, vertically group the assessment information top-down according to tier level within the overall system architecture.

2. **Subsystem**

   Horizontally group the assessment information about each tier according to subsystem (e.g., first by parent subsystem and then alphabetically by subsystem name)

3. **Quality Factor**

   Group the assessment information about each subsystem according to the quality factors that are relevant to the subsystem, specifically, by the *claim* that the subsystem architecture adequately supports the achievement of the quality factor (e.g., performance and usability).

4. **Quality Subfactor**

   Group the assessment information of the quality factors according to quality subfactor, specifically, by the claims that the subsystem architecture adequately supports the achievement of the quality subfactors.

5. **Quality Case**

   Group the assessment information of the quality subfactors according to individual quality case.

   a. **Claims**
      Finally, group the claims first by quality factor goal claims, second by quality subfactor goal claims, and third by quality requirements claims.
   b. **Arguments**
      Similarly, group the arguments by architectural decisions and associated rationales.

c. Evidence

Similarly, group the evidence first by documentation (e.g., diagram, model, and document) and by witnessed demonstrations.

Note that with this approach, there is typically one quality case per quality subfactor. There could also possibly be one quality case per quality factor if the quality factor does not have multiple, obvious quality subfactors. It is conceivable that one could develop a single compound quality case for each subsystem by combining all of the quality cases for all of the relevant quality factors for that subsystem.[25] However, such a combined quality case would be very large and difficult to understand and navigate. Similarly, one could produce a compound quality case for all subsystems in a tier of the architecture or even a single, mega-quality case for the entire system by combining the lower level quality cases. Although possible in theory, this is inadvisable in practice as it becomes much too large and complex for a human to readily comprehend.

As illustrated in Figure 8, a single quality case can be thought of as a pyramid pointing to the architects' assertion that the system and subsystems architecture helps the system or subsystems being assessed possess an adequate amount of some type of quality. This four-layer pyramid consists of

1. **Top-Level Claims**

   Top-level claims are claims that "the system/subsystem architecture helps the system/subsystem achieve one or more quality factor or quality subfactor goals." These are made verifiable by second-level claims.

2. **Second-Level Claims**

   Second-level claims are claims that "the system/subsystem architecture helps the system/subsystem meet its associated quality requirements." The belief in which is justified by arguments.

3. **Arguments**

   Arguments consist of architecture decisions with associated rationales. Arguments are supported on a strong foundation of evidence.

4. **Evidence**

   Evidence consists of architectural documentation and demonstrations witnessed by the assessors.

[25] A quality factor is considered relevant to a system/subsystem if it is architecturally significant and if it has been selected as being within scope of the assessments.

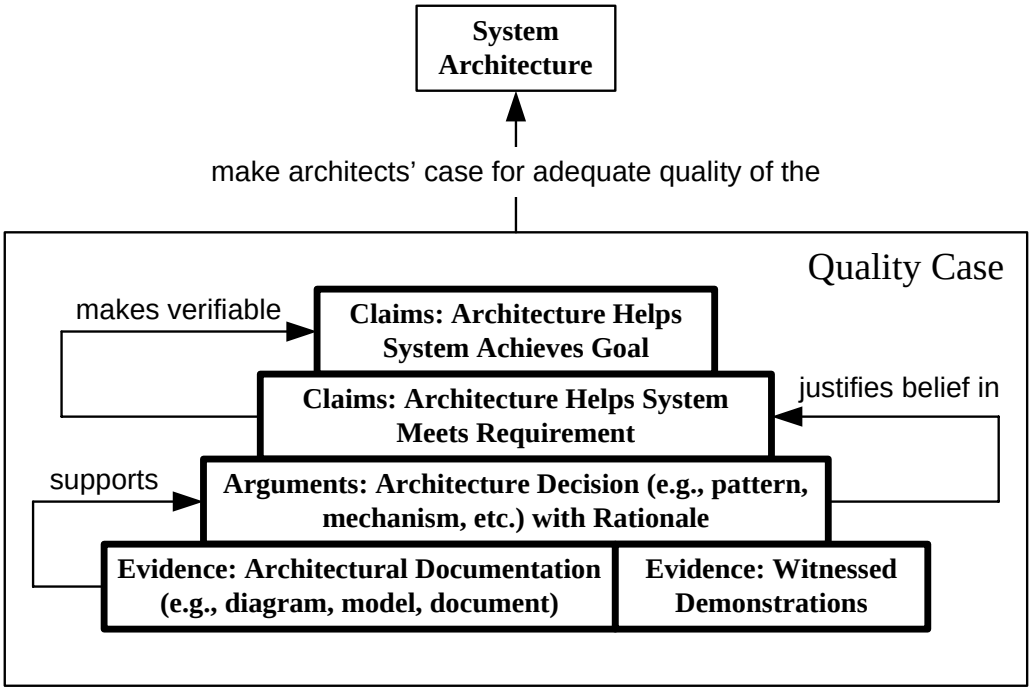

Figure 8: Layered Structure of Quality Cases

The pyramid of information composing a quality case can be quite complex. Although it consists of a relatively small number of claims, it can include a sizable number of arguments and an even larger amount of evidence (hence the pyramid shape). This large amount of information can be overwhelming, especially to the assessors who will not be as familiar with it as the architects and especially if it is presented to them only in textual form (not counting any evidence in the form of diagrams). Thus, a diagram summarizing the actual content of a quality case would be useful as an introduction of the quality case and as an aid in navigation through its component information.

But what are the characteristics of such a diagram? Clearly, a quality case diagram and associated notation should

- help the stakeholders (i.e., architects, assessors, and requirements engineers) navigate through the potentially quite large amount of information composing a quality case
- summarize the component information of a quality case so that it helps the stakeholders manage the complexity of the quality case
- clearly differentiate the different types of component information including
  - claims
    - quality factor goal
    - quality subfactor goals
    - quality requirements goals
  - arguments
    - architecture decisions
    - associated rationales

  - evidence
    - architecture documentation (e.g., diagrams, models, and documents)
    - witnessed demonstrations
- summarize the layered structure of a quality case including the important relationships between the components
- use a standard notation (e.g., Unified Modeling Language [UML]) that is
  - easy to learn or else is widely known by stakeholders
  - intuitively understandable (e.g., not use large numbers of arbitrary symbols)
  - easy to draw on a single chalk board or white board
  - not required to have sophisticated tool support
  - is nevertheless supported by commonly available tools for inclusion in presentation materials (and possibly in deliverable quality case documentation if so desired)
- be practical in the sense of being both
  - useful on real projects
  - scalable to the size of real quality cases

Unfortunately, the existing diagramming method (Goal Structuring Notation [Weinstock 04]) fails to exhibit many of the positive characteristics in the preceding list. Therefore, this handbook introduces Quality Case Diagrams, a specialized UML class diagram specifically designed to exhibit these positive characteristics. The quality case diagram notation consists of

- class icons (i.e., rectangles) to model the component parts of a quality case
- UML standard symbols for stereotypes (i.e., the left and right angle quotes: "«" and "»") to signify the different types of components (i.e., claims, arguments, and evidence)
- explicit labeling (e.g., "Goal:", "Requirement:") to clearly differentiate different kinds of claims
- UML standard aggregation symbol (i.e., small diamond) to indicate aggregation
- unidirectional labeled associations to model the remaining relationships

Figure 9 is an example quality case diagram summarizing the architecture's support for interoperability. Note that the class icons are placed in horizontal layers to help clarify the directions of the dependency relationships and to group similar component types. Note also that

- Belief in a single claim can be justified by multiple arguments.
- A single argument can justify belief in multiple claims.
- A single argument can be supported by multiple pieces of evidence.
- A single piece of evidence can support multiple arguments.

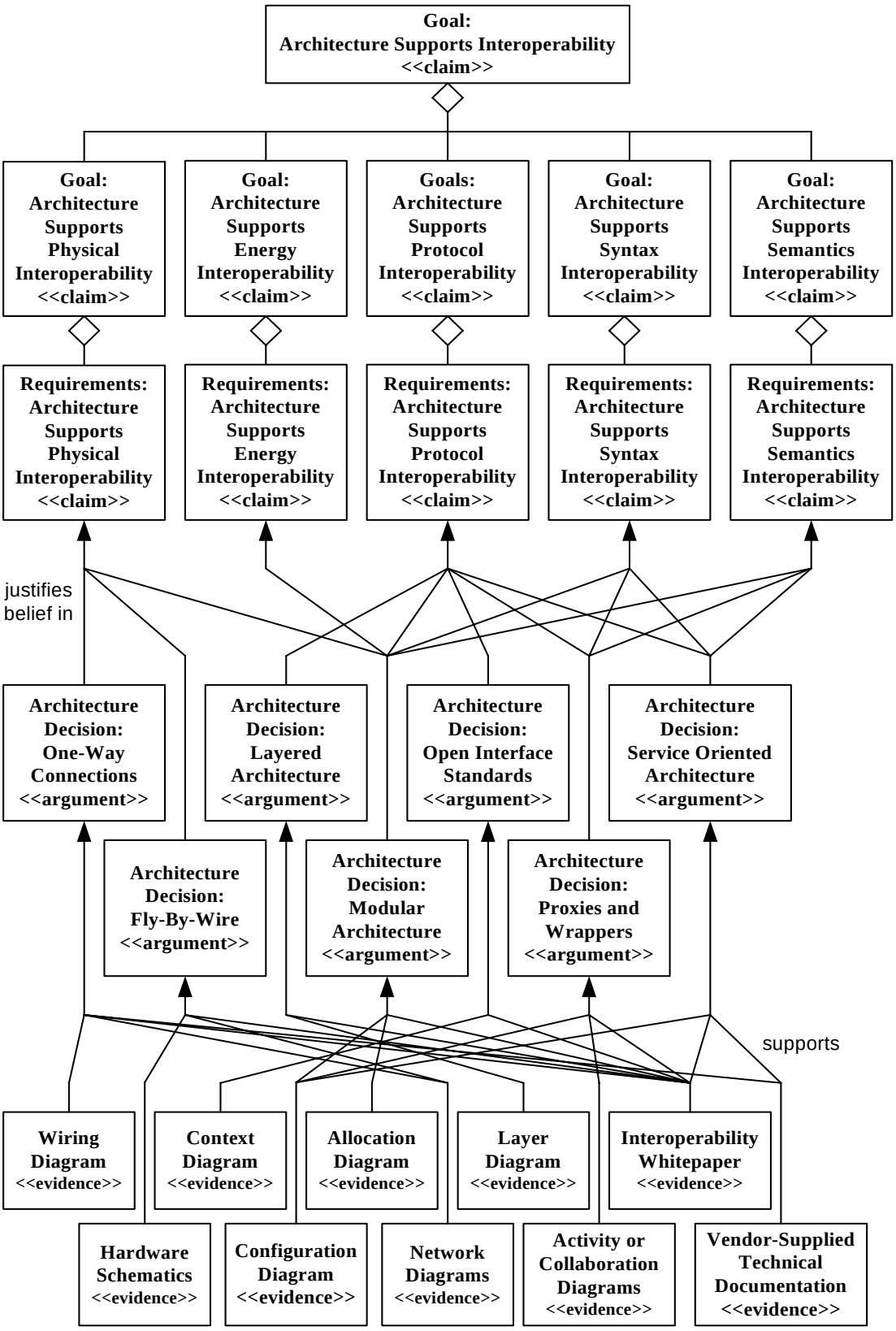

Figure 9: Example Quality Case Diagram

To make quality case diagrams more understandable, it is useful to order the nodes within a single horizontal level of the diagram as follows:

- alphabetically left to right (e.g., quality subfactor claims)
- in decreasing order of importance (e.g., arguments – architectural decisions)
- by node sizes in order to fit nicely when having multiple rows of arguments and evidence
- to minimize the crossing of lines (usually the most important overriding criteria)

3.3 Potential Concerns

By now, it should be clear that safety cases can be generalized to quality cases to handle all of the different quality factors and quality subfactors. It should also be clear that quality cases can occur earlier in the development process, assessing architecture quality versus assessing system quality. However, there are several significant differences between traditional safety cases and architectural quality cases. One could argue that these differences are so significant that it is inappropriate to use quality cases to assess the quality of system and subsystem architectures or that one should modify the QUASAR method or architectural quality cases to address any limitations in their use.

The following subsections provide a brief analysis of these differences and their ramifications.

3.3.1 Use *All* Quality Factors

Differences: Safety cases and QUASAR architecture cases exhibit the following differences in terms of quality factors that are covered:

- Safety cases are naturally restricted to safety, which is only one of a large number of quality factors.
- QUASAR architecture cases are a kind of quality case. As such, they can be developed for any quality factor or quality subfactor, not just safety.

Ramifications: All quality factors can have associated quality goals and requirements, so the architects can make associated claims for any quality factor (i.e., that the architecture adequately supports the system or subsystem to achieve its associated goals and meet its associated requirements). Similarly, regardless of the quality factor, the architects make architectural decisions that should have associated rationales. Thus, the architects should be able to make clear and compelling arguments as to why their architecture justifies belief in their claims. Finally, regardless of the quality factor, architects need to document their architectural decisions in diagrams, models, and documentation, enabling them to provide sufficient evidence that supports their arguments. Therefore, the type of quality factor does not negatively impact the architects' ability to create quality cases consisting of claims, justifying arguments, and supporting evidence. In fact, non-safety quality cases should typically be

smaller and simpler than safety cases because (1) less information is required because sufficient information to justify formal safety certifications is unnecessary and (2) the scope is restricted to only architectural information.

Not restricting QUASAR architecture cases to only the safety quality factor has a *positive impact* on the QUASAR method.

3.3.2 All Quality Factors Not Equally Important

Differences: System quality cases and QUASAR architecture cases exhibit the following differences in terms of importance:

- Safety cases only address system safety and are only developed for safety-critical systems when safe system usage is critically important.
- QUASAR architecture cases - The absolute and relative importance different quality factors varies from system to system. Although safety is crucial in safety-critical systems, availability and reliability may be much more important on some systems, while security might be paramount in military systems storing classified data. QUASAR architecture cases can be developed for any quality factor or quality subfactor.

Ramifications: The assessors and architects have the flexibility to select, develop, and assess quality cases only for those quality factors and subfactors for which quality cases are justified in terms of cost and risk.

Using QUASAR architecture cases to address only those quality factors associated with the highest risk or system cost has the *positive impact* of significantly lowering project risk and potentially lowering system development cost and maintaining the schedule.

3.3.3 Use for Demonstration (Certification) vs. Assessment

Differences: Safety cases and QUASAR architecture cases exhibit the following differences in terms of demonstration versus assessment:

- Safety cases have historically been used during safety certification to demonstrate that a system is sufficiently safe to use. Safety cases have often been used to demonstrate the safety of public transportation systems (e.g., airplanes, trains, and APMs) to ensure public safety.
- QUASAR architecture cases are primarily used to *assess* the quality of system *architectures*, not to *demonstrate* the quality of the *system*.

Ramifications: Because safety certification is contractually or legally mandated, it justifies a very heavy investment in expense in both time and money to officially demonstrate. Whereas safety certification demands a demonstration (i.e., proof) of safety, architecture assessment typically only requires a preponderance of evidence to pass. Architecture quality cases will typically be significantly smaller than safety cases.

Using QUASAR architecture cases for assessment rather than demonstration and certification has the *positive impact* of significantly lowering their cost, both in terms of effort and schedule.

3.3.4 System Quality Cases vs. QUASAR Architecture Cases

Differences: System quality cases and QUASAR architecture cases exhibit the following differences in terms of system versus architecture cases:

- System quality cases present the developers' claims, arguments, and evidence that the system has sufficient quality.
- QUASAR architecture cases present the architects' claims, arguments, and evidence that the architecture adequately supports the quality of the system or subsystem.

Ramifications: Because architecture quality cases are restricted to architecture claims, architecture arguments, and architecture evidence, they do not contain design, implementation, integration, or test information. Architecture quality cases also only address requirements to the extent to which the architecture must meet architecturally significant requirements. Compared to system quality cases, architecture quality cases will therefore be significantly smaller and significantly less costly in terms of both effort and schedule.

Using QUASAR architecture cases rather than system quality cases has the *positive impact* of significantly lowering their cost, both in terms of effort and schedule.

3.3.5 During Development vs. End of Development

Differences: Safety cases and QUASAR architecture cases exhibit the following differences in terms of timing:

- Safety cases – Because safety is a system property, it can be demonstrated and certified only at the end of system development. To demonstrate that a system is safe, one must demonstrate proper requirements, architecture, design, implementation, integration, and test because a failure in any of these disciplines can result in an unsafe system. Although safety cases can and should be started early during development, this is why they are completed only at the end of system development.
- QUASAR architecture cases – The architecture of the system is developed in an incremental, iterative, and parallel manner as the system's subsystems are identified and developed. Because the architecture drives the design, implementation, integration, and integration testing, the architecture of the individual subsystems is developed relatively early during development.

Ramifications: Architecture cases are produced and reviewed relatively early during the development cycle as the architecture is developed. Architecture cases are also developed and reviewed in an incremental, iterative, and parallel manner as the architectures of the system's subsystems are produced.

Using QUASAR architecture cases early in the development cycle has a positive impact of supporting the assessment of the architecture earlier during development than when using complete quality cases, which must be used at the end of development because they address design, implementation, integration, and testing as well as requirements and architecture. This enables problems to be identified and corrected earlier, when they are less expensive and more likely to be properly addressed.

4 QUASAR Teams

The following teams collaborate to prepare for and perform the architecture assessments:

1. **Assessment Team**

   One or more assessment teams

   a. collaborate with the architecture team to set the scope of the assessments
   b. independently assess the quality of the system or subsystem architecture produced by the associated architecture team

2. **Architecture Team**

   The architecture teams at the different tiers in the system hierarchy

   a. collaborate with the assessment team to set the scope of the assessments
   b. make their *quality cases* to the assessment teams consisting of
      – *claims* that their architecture fulfills its associated architecturally significant requirements
      – clear and compelling *arguments* supporting their claims (e.g., describe the associated architectural decisions they have made and the rationales for these decisions)
      – adequate official *evidence* (e.g., architectural diagrams, models, and documents) backing up their arguments

3. **Requirements Team**

   The requirements teams at the different tiers in the system hierarchy

   a. engineer the architecturally significant quality requirements
   b. collaborate with the assessment team and architecture team to set the scope of the assessments

Note that the number of teams and their membership varies from program to program depending on many factors such as the size and complexity of the system and the [contractual] relationships between the assessment teams and the architecture teams. Although it is typically important for assessment teams to remain independent from architecture teams to ensure that the assessment of the architecture is objective, in practice, a person can be a member of multiple teams. Thus, the same person may be a member of multiple assessment teams, of multiple architecture teams, or of both the requirements team and architecture team.

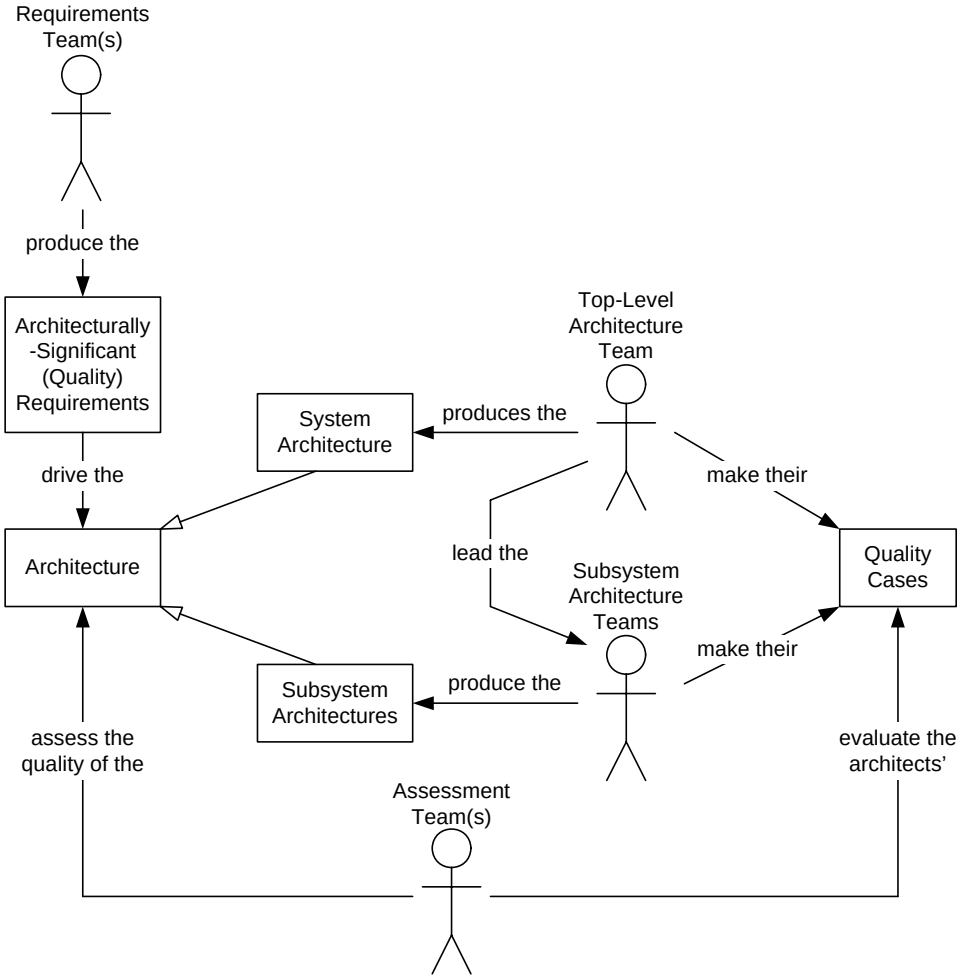

Figure 10: Teams and Their Interactions

4.1 Assessment Team

The following definition, responsibilities, membership, and work products apply to the assessment team. On large programs, multiple assessment teams may perform assessments in parallel.

- **Definition**

  The assessment team independently[26] assesses the quality of the architectures of a system and its subsystems. This team assesses the degree to which architectures support the derived requirements that have been allocated to the subsystems and that have driven the development of the architectures.

[26] To obtain an objective assessment, it is important that the assessment team is independent of the architecture teams (e.g., in terms of staffing and reporting).

- **Responsibilities**

  The members of the assessment team typically share the following responsibilities:

  - **Include subject matter experts.**
    If appropriate, the assessment team should identify and invite special subject matter experts to join them. These experts in the application domain or a relevant specialty engineering field can understand specialized architectural information, the architects' decisions, and their associated rationales.
  - **Determine assessment scope.**
    The assessment team works with the architecture teams to set the scope of the assessments in terms of the architectural elements assessed and the relevant quality factors.
  - **Collaborate.**
    The assessment team collaborates with the subsystem requirements and subsystem architecture teams to schedule the meetings.
  - **Properly conduct preparations.**
    Members of the assessment team prepare for meetings by reading preparatory information supplied by the requirements and architecture teams.
  - **Assess understanding.**
    Members of the assessment team evaluate the architects' understanding of the quality requirements that drive their architectures.
  - **Assess quality cases.**
    Members of the assessment team evaluate the architects' claims, arguments, and evidence concerning their architectures' support for derived and allocated quality requirements.
  - **Ask questions.**
    Members of the assessment team ask probing questions concerning the quality requirements and the architecture. Note that is important that members of the assessment team have strong backgrounds, be persistent in eliciting satisfactory answers from the system and subsystem architects, and not be easily intimidated by such senior members of the development staff.
  - **Publish results.**
    Assigned members of the assessment team prepare and distribute the meeting minutes and reports.
  - **Update method.**
    Assigned members of the assessment team update the assessment method as appropriate based on lessons learned during the assessments.

- **Membership**

  The members of the assessment team typically include people filling one or more of the following roles:

  - **Assessors**
    Assessors are responsible for technically assessing the architectural information provided and presented by the architecture team to determine if the architecture suffi-

ciently supports its allocated and derived quality requirements. This role can include customer representatives and consultants who are also architects. The assessment team should retain a consistent core set of assessors who take part in all (or almost all) subsystem assessments.

  - **Assessment Team Leader**
    The assessment team leader is an assessor who leads the assessment team. This role often includes the responsibility of performing final negotiations with the architecture teams regarding issues for which consensus is difficult to obtain.
  - **Meeting Facilitator**
    The meeting facilitator is an assessor who facilitates during meetings, ensuring that the meeting discussions stay on track and that the meeting stays on schedule.
  - **Subsystem Liaison**
    The subsystem liaison is the member of the customer (acquisition) organization who oversees the development of a subsystem. The subsystem liaison typically works closely with the subsystem development team (and therefore the subsystem architecture team) and should therefore be well versed in the subsystem architecture and its status. Note that the subsystem liaison is typically not a core member of the assessment team unless only a small number of subsystems are being assessed and the core team contains all subsystem liaisons.
  - **Subject Matter Experts (SMEs)**
    Subject matter experts are assessors who are also specialty engineering experts.[27] They are responsible for providing expertise in the
    - application domain of the subsystem being assessed (e.g., avionics specialists for avionics subsystems or communications engineers for communications subsystems)
    - quality factor (e.g., reliability engineers, safety engineers, and security engineers)
  - **Scribe**
    The scribe is an assessor who captures general observations and action items such as requests for information or requests for action during the meeting.

- **Work Products**
  - System Architecture Assessment Initiation Phase Work Products
    - Architecture Assessment Training Materials
    - Architecture Assessment Procedure
    - Initial Kickoff Meeting Agenda
    - Initial Kickoff Meeting Notes (developed by individual attendees)
    - Architecture Assessment Schedule
    - Architecture Assessment Action Item List

[27] SMEs typically take part in the assessments on an as-needed basis. Thus, they are typically not a part of the set of core members of the assessment team. It should be decided early as to whether or not SMEs may vote on the subsystem assessment results (i.e., whether or not the subsystem architecture adequately supports a specific class of quality requirements and therefore what color it earns on the subsystem architecture support matrix). Refer to Section 6.3.6 for more information about the subsystem architecture support matrix and its contents.

  - Initial Kickoff Meeting Minutes (built using individual attendee's notes)
- Subsystem Requirements Meeting Work Products
  - Subsystem Requirements Meeting Checklist
  - Subsystem Requirements Meeting Assessor Notes (individual assessor's notes)
  - Subsystem Requirements Meeting Outbrief
  - Subsystem Requirements Meeting Minutes (built using individual assessor's notes)
  - Updated Assessment Action Item List
- Subsystem Architecture Assessment Meeting Work Products
  - Subsystem Architecture Assessment Checklist
  - Subsystem Architecture Assessment Meeting Assessor Notes (individual assessor's notes)
  - Subsystem Architecture Support Matrix
  - Subsystem Architecture Assessment Meeting Outbrief
  - Subsystem Architecture Assessment Meeting Report (built using individual assessor's notes)
  - Updated Assessment Action Item List

4.2 Architecture Teams

Two types of architecture teams are involved in the architecture assessment method: the top-level architecture team and multiple subsystem architecture teams. The same people may fill roles on both types of architecture teams, especially on smaller projects. On large projects, multiple subsystem architecture teams will probably participate in parallel assessments.

The following definition, responsibilities, membership, and work products apply to the architecture teams.

4.2.1 Top-Level Architecture Team

- **Definition**

  The top-level architecture team is the team that produces the top-level architecture being assessed.

- **Responsibilities**
  - Lead the lower level (subsystem) architecture teams.
  - Understand the assessment procedure and share this knowledge with the subsystem architecture teams.
  - Attend the initial kickoff meeting.
  - Collaborate with the assessment team and top-level requirements team to
    - Tailor the architecture assessment method.
    - Set the scope of the architecture assessments in terms of subsystems, relevant types of architecturally significant requirements (e.g., quality requirements), and level of resources (e.g., time and money) to be invested in the assessments.
    - Develop an initial schedule for assessments.

- Work with the subsystem architecture teams to ensure that they understand the system architecture assessment method and their responsibilities with regard to its application.

- **Membership**
  - lead system architect
  - subsystem architects
- **Work Products**
  - Initial Kickoff Meeting Work Products
    - recommended tailoring of the architecture assessment method
    - recommendations regarding the scope of the architecture assessments
    - recommendations regarding the scheduling of assessment meetings
  - Subsystem Requirements Meeting Work Products
    - none
  - Subsystem Architecture Assessment Meeting Work Products
    - none

4.2.2 Subsystem Architecture Teams

- **Definition**

  Subsystem architecture teams are the teams that produce the individual subsystem architectures being assessed.

- **Responsibilities**
  - Prepare and provide preparatory materials for the meetings to the assessment team sufficiently early for the assessment team to properly review them.
  - Prepare and present meeting presentation materials to the assessment team to convince them that the subsystem adequately supports its allocated and derived quality requirements. This includes quality cases consisting of
    - *claims* that their architecture provides sufficiently support for the derived architecturally significant requirements that have been allocated to their subsystem
    - clear and compelling *arguments* supporting these claims
    - sufficient *evidence* backing up the arguments
  - Answer evaluators' questions regarding their architectural decisions.
  - Review subsystem meeting outbriefs, minutes, and reports for factual errors (e.g., incorrect observations and missing evidence).
- **Membership**
  - subsystem architects
  - lead architect (if appropriate during the meetings)
  - sub-subsystem architect(s)
  - specialty engineering team representatives (for quality factors treated as specialty engineering, such as reliability, safety, and security)

- **Work Products**
  - Initial Kickoff Meeting Work Products
    - none
  - Subsystem Requirements Meeting Work Products
    - Subsystem Requirements Review Preparatory Materials (architecture-related information)
    - Subsystem Requirements Review Presentation Materials (architecture-related information)
    - Subsystem Requirements Review Meeting Agenda (architecture-related information)
  - Subsystem Architecture Assessment Meeting Work Products
    - Subsystem Architecture Preparatory Materials
    - Subsystem Architecture Meeting Agenda
    - Subsystem Architecture Presentation Materials

4.3 Requirements Teams

Two types of requirements teams are involved in the architecture assessment method: the top-level requirements team and multiple subsystem requirements teams. The same people may fill roles on both types of requirements teams, especially on smaller projects. On large projects, multiple subsystem requirements teams will probably participate in parallel assessments.

The following definition, responsibilities, membership, and work products apply to the requirements teams.

4.3.1 Top-Level Requirements Team

- **Definition**

  The top-level requirements team is the team that engineers (e.g., identifies, analyzes, specifies, and manages) the system requirements of the system architecture being assessed.

- **Responsibilities**
  - Engineer system goals and requirements.
    - Engineer system functional, data, and interface requirements and appropriate constraints.
    - Engineer system quality requirements.[28]
  - Ensure that all requirements (especially the architecturally significant requirements) are cohesive, complete, consistent, correct, current, externally observable, feasible,

[28] In practice, many system-level requirements engineers are primarily trained in methods (e.g., use case modeling) for engineering functional requirements. Unfortunately, many do not know how to engineer quality requirements that are feasible, unambiguous, and verifiable.

mandatory, relevant, stakeholder-oriented, and unambiguous, can be validated and verified.
  - Lead the lower level (subsystem) requirements teams.
  - Understand the assessment procedure and share this knowledge with the subsystem requirements teams.
  - Attend the initial kickoff meeting.

  - Collaborate with the assessment team and top-level architecture team to
    - Tailor the architecture assessment method.
    - Set the scope of the relevant types of architecturally significant requirements (e.g., quality requirements), and level of resources (e.g., time and money) to be invested in the assessments.
    - Develop an initial schedule for assessments.
  - Work with the subsystem requirements teams to ensure that they understand the system architecture assessment method and their responsibilities with regard to its application.

- **Membership**
  - requirements team leader
  - requirements engineers
  - specialty engineering team representatives (for quality factors treated as specialty engineering, such as reliability, safety, and security)
- **Work Products**
  - System Architecture Assessment Initiation Meeting Work Products:
    - Initial Kickoff Meeting Notes

4.3.2 Subsystem Requirements Teams

- **Definition**

  The subsystem requirements teams are teams that engineer (e.g., identify, analyze, specify, and manage) the derived requirements that have been allocated to the individual subsystem architectures being assessed

- **Responsibilities**
  - Engineer derived goals and requirements.
    - Engineer subsystem functional, data, and interface requirements and appropriate constraints.
    - Engineer quality requirements.[29]
  - Derive subsystem goals and requirements.

[29] In practice, subsystem requirements engineers are like system requirements engineers in that they are primarily trained in methods (e.g., use case modeling) for engineering functional requirements. Unfortunately, many do not know how to engineer quality requirements that are feasible, unambiguous, and verifiable.

- Ensure that all requirements (especially the architecturally significant requirements) are cohesive, complete, consistent, correct, current, externally observable, feasible, mandatory, relevant, stakeholder oriented, unambiguous, and can be validated and verified.
- Collaborate with the subsystem architecture team to ensure that all architecturally significant requirements are properly
  - derived
  - identified (e.g., tagged) as such
  - specified in sufficient detail to provide adequate guidance

- **Membership**
  - requirements team leader
  - requirements engineer
  - specialty engineering team representatives (for quality factors treated as specialty engineering, such as reliability, safety, and security)
- **Work Products**
  - System Architecture Assessment Initiation Meeting Work Products:
    - none
  - Subsystem Requirements Meeting Work Products:
    - Subsystem Requirements Meeting Preparatory Materials
    - Subsystem Requirements Meeting Presentation Materials
    - Subsystem Requirements Trace
  - Subsystem Architecture Assessment Meeting Work Products
    - none

5 QUASAR Phases and Tasks

As illustrated in Figure 11, the QUASAR method is decomposed into the following phases:

1. **System Architecture Assessment Initiation Phase**
   This preparatory phase occurs once at the beginning of the overall system assessment.

   Subsystem Phases

   a. **Subsystem Requirements Review Phase**
      This phase is repeated for each subsystem for which the architecture is being assessed. It ensures that the associated quality requirements are properly allocated and specified. It also ensures that the architecture team is ready to develop their quality cases.

   b. **Subsystem Architecture Assessment Phase**
      This phase is repeated for each subsystem for which the architecture is being assessed. During this phase, the subsystem architecture team presents their quality cases to the assessment team.

2. **System Architecture Assessment Summary Phase**
   This phase typically occurs once at the end of the overall assessment. During this final phase, the results of the subsystem architecture quality assessments are summarized and presented to their stakeholders.

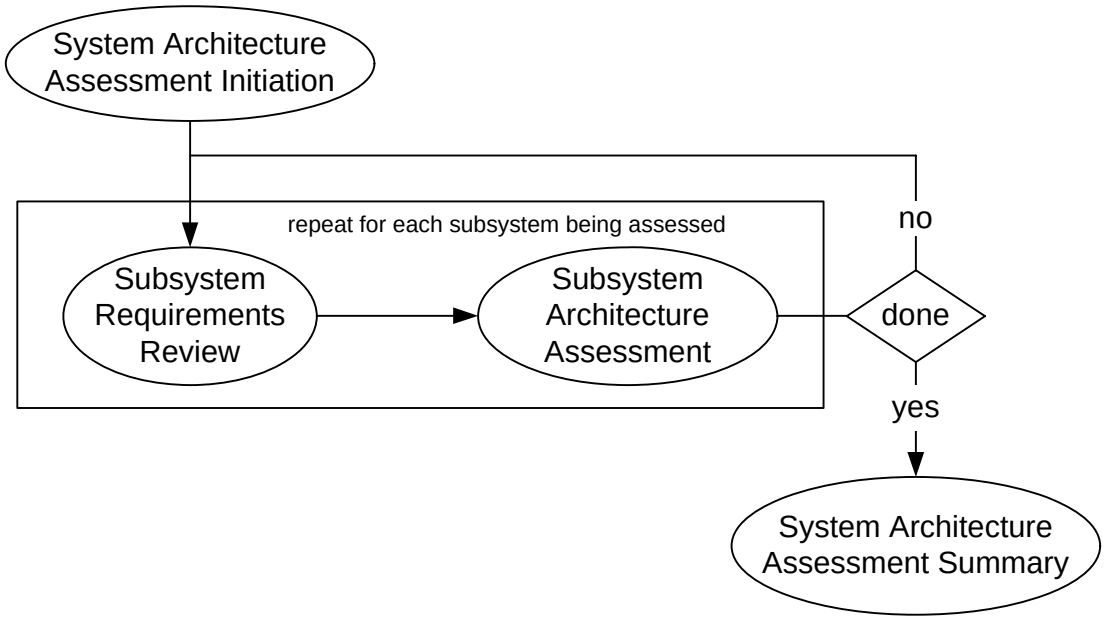

Figure 11: QUASAR Phases

As illustrated in Figure 12, each of these phases is decomposed into three subtasks: pre-meeting preparation, the associated meeting, and post-meeting follow-through. The figure does not show any iteration that might be necessary.[30]

1. System Architecture Assessment Initiation Phase

a. Task: System Architecture Assessment Initiation – Preparation

The assessment team and overall architecture team prepare by exchanging and reading relevant pre-meeting documentation.

b. Task: System Architecture Assessment Initiation – Meeting

The assessment team and overall architecture team hold an initial kickoff meeting during which they agree on the scope and method for the architecture assessments.

c. Task: System Architecture Assessment Initiation – Follow-Through

The assessment team reports their findings and recommendations to the overall architecture team and other stakeholders.

Subsystem Assessments

For *each* subsystem being assessed on a subsystem by subsystem basis perform the following tasks:

a. Subsystem Requirements Review Phase

– **Task: Subsystem Requirements Review – Preparation**
The assessment team and subsystem architecture team prepare by exchanging and reading relevant pre-meeting documentation.

– **Task: Subsystem Requirements Review – Meeting**
The subsystem architecture team demonstrates to the assessment team their knowledge of the architecturally significant requirements and their understanding of what is expected of them during the coming assessment.

– **Task: Subsystem Requirements Review – Follow-Through**
The assessment team reports their findings and recommendations to the subsystem architecture team and other stakeholders.

b. Subsystem Architecture Assessment Phase

– **Task: Subsystem Architecture Assessment – Preparation**
The assessment team prepares by exchanging and reading relevant pre-meeting documentation provided by the subsystem architecture team.

– **Task: Subsystem Architecture Assessment – Meeting**
The subsystem architecture team presents to the assessment team their cases that the architecture adequately supports its architecturally significant requirements.

– **Task: Subsystem Architecture Assessment – Follow-Through**
The assessment team reports their findings and recommendations to the subsystem architecture team and other stakeholders.

[30] The two major reasons for repeating the requirements reviews and architecture assessments are (1) if the subsystems were not ready or (2) if the reviews and assessments did not (completely) pass the first time.

2. **System Architecture Assessment Summary Phase**

   a. **Task: System Architecture Assessment Summary – Preparation**

   The assessment team prepares by rolling up the results of the individual subsystem assessments into an overall system architecture assessment.

   b. **Task: System Architecture Assessment Summary – Meeting**

   The assessment team reports the overall system architecture assessment results to the system architecture team and other stakeholders.

   c. **Task: System Architecture Assessment Summary – Follow-Through**

   The assessment team captures lessons learned and updates the architecture assessment method.

The subsections in this section discuss these phases in more detail.

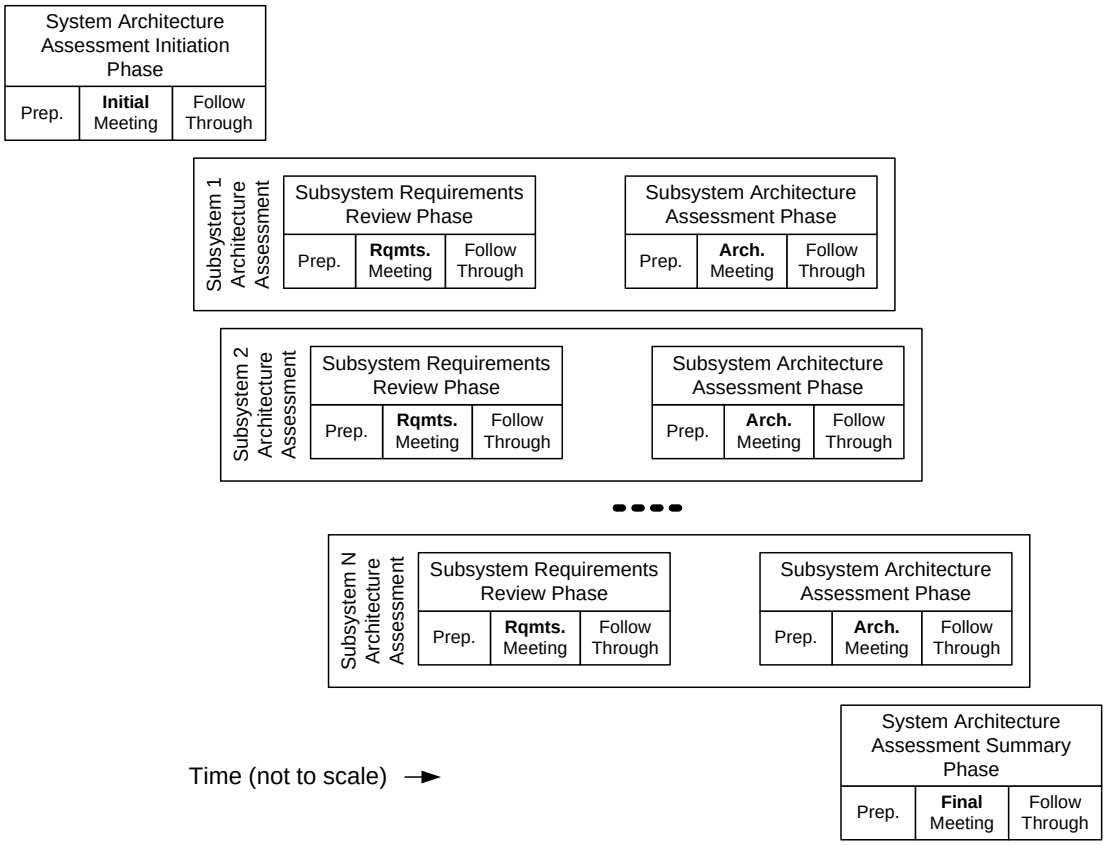

Figure 12: QUASAR Phases and Tasks

5.1 System Architecture Assessment Initiation Phase

A system architecture assessment initiation phase occurs at the beginning of the QUASAR architecture assessment method.

Phase Objective

The objective of this phase is to properly prepare both the assessment and architecture teams to assess the subsystem architectures.

Phase Tasks

As illustrated in Figure 12, this assessment initiation phase consists of the following three tasks, which are performed sequentially:

1. **System Architecture Assessment Initiation – Preparation**
2. **System Architecture Assessment Initiation – Initial Kickoff Meeting**
3. **System Architecture Assessment Initiation – Follow-Through**

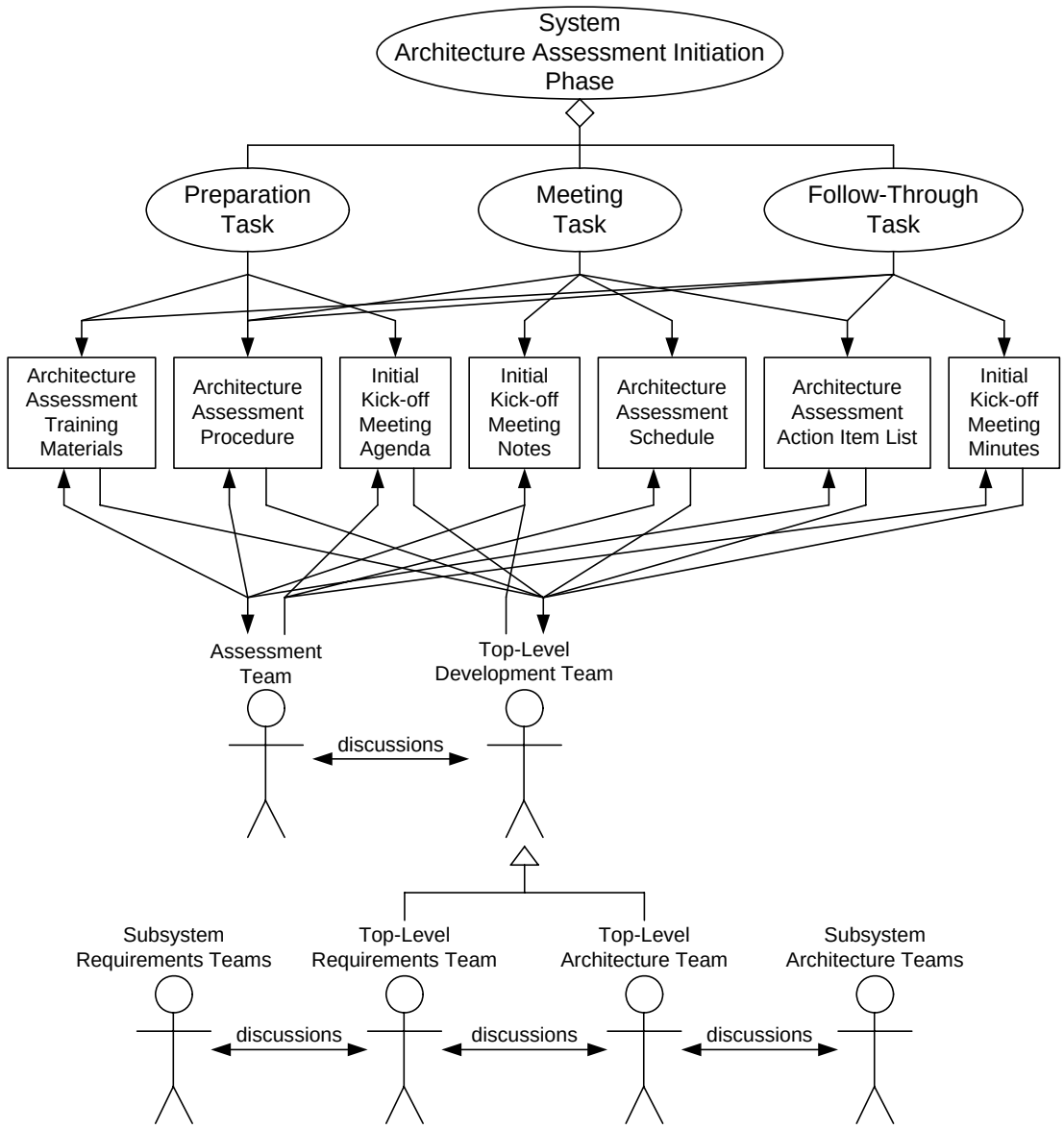

Figure 13: System Architecture Assessment Initiation Phase

As illustrated in Figure 13, three tasks of the system architecture assessment initiation phase involve the assessment team and the top-level architecture team producing or using following work products:

1. **Architecture Assessment Procedure**
2. **Architecture Assessment Training Materials**
3. **Initial Kickoff Meeting Agenda**
4. **Initial Kickoff Meeting Notes**
5. **Initial Kickoff Meeting Minutes**
6. **Assessment Schedule**
7. **Assessment Action Item List**

5.1.1 System Architecture Assessment Initiation – Preparation

Preparation Task Objective
The objective of this task is to ensure that the system architecture team and the assessment team are properly prepared for the assessment kickoff meeting.

Preparation Task Duration
The duration of this task largely depends on the availability of the members of the system architecture team and the members of the assessment team; it may vary anywhere from a few days to a few weeks.

Preparation Task Preconditions
This task can be started when the decision to perform a system architecture quality assessment has been made.

Preparation Task Steps
Prior to the initial kickoff meeting, perform the following steps:[31]

1. **Staff assessment team.**

   Step: The management of the assessment organization ensures that the assessment team is properly staffed with its initial members.[32]

   Rationale: As documented in Section 4.1, the assessment team needs to be staffed with an assessment team leader, a meeting facilitator, assessors, subject matter experts, and a scribe. Although all team members may not be available at the beginning of the process, and although subject matter experts will be members of the assessment team based on availability and relevance to the subsystem being assessed, a core group of permanent members should be assigned to the team at this time.

2. **Train assessment team.**

   Step: A qualified and experienced member of the assessment team uses the architecture assessment training materials and architecture assessment procedure to train the other members of the assessment team in the proper use of the architecture assessment method.

   Rationale: The assessment team will be much more effective if they understand the method that they will use.

---

[31] The amount of time prior to the meeting will vary depending on the circumstances of the assessment. Care should be taken to enable just-in-time preparation, while ensuring adequate preparation time.

[32] The tasks, subtasks, and steps are numbered in this handbook for the sake of easy identification during discussions and negotiations between teams. Because an iterative, incremental, and parallel development cycle and corresponding assessment method is assumed, these tasks, subtasks, and steps may be performed in different orders and concurrently.

3. **Identify top-level requirements and architecture teams.**

   Step: The assessment team works with the management of the development organization to identify their top-level requirements team, their top-level architecture team, and the members of these two teams who will participate in the assessment.

   Rationale:

   – Depending on the timing of the system architecture assessment initiation meeting, the members of these two teams should know primary architecturally significant system requirements or at least the associated quality goals if proper requirements have not yet been specified. The system requirements team should have by this time elicited these architecturally significant requirements from the system's various stakeholders. This knowledge will be necessary to set the scope of the overall system architecture quality assessment.
   – The top-level requirements team is typically responsible for engineering the requirements of either the overall system or the top-level subsystem, the architecture of which is being assessed. They are therefore responsible for engineering the architecturally significant system quality requirements.
   – Similarly, the top-level architecture team is typically responsible for developing the architecture of either the overall system or the top-level subsystem, the architecture of which is being assessed. They will also be responsible for ensuring the architectural integrity of the lower level subsystems.
   – Depending on the structure of the development organization, the members of these two teams may be difficult to identify.

4. **Train top-level architecture team.**

   Step: Several weeks prior to the initial kickoff meeting, a qualified and experienced member of the assessment team uses the architecture assessment training materials and architecture assessment procedure to train the members of the top-level requirements and architecture teams in the proper use of the architecture assessment method. The members of the top-level requirements and architecture teams are expected to read the architecture assessment procedure.

   Rationale:

   – Training must typically be provided several weeks prior to the initial kickoff meeting because the top-level architecture team members need adequate time to read the assessment procedure in addition to preparing for the assessment.
   – The members of the top-level architecture team need to understand what is expected, both of them and of the members of the lower level subsystem requirements and architecture teams, to ensure that the assessment method meets the needs of the requirements and architecture teams as well as the needs of the assessment team.

5. **Organize the meeting.**

   Step: At least two weeks prior to the initiation meeting, the assessment, system requirements, and system architecture teams collaborate to

a. *Identify meeting attendees and other stakeholders.*

Substep: The assessment, system requirements, and system architecture teams collaborate to develop a list of meeting attendees and other stakeholders.
Rationale: It is important to ensure that no significant stakeholder is overlooked.

b. *Set time and location.*

Substep: The assessment, system requirements, and system architecture teams collaborate to set an exact time and location (e.g., address, building number, and room number) for the system architecture assessment initiation meeting.
Rationale: Attendees need an exact time to avoid schedule conflicts.

c. *Develop meeting agenda.*

Substep: The assessment, system requirements, and system architecture teams collaborate to produce a meeting agenda covering setting the assessment scope and schedule, tailoring the assessment method, and meeting wrap-up (e.g., assignment of action items).
Rationale: Agendas enable attendees to identify the parts of the meeting that are most important to them.

d. *Invite stakeholders.*

Substep: The assessment team sends an invitation including the meeting agenda to the identified meeting attendees and other stakeholders.
Rationale: Documented invitations (e.g., email with attached agenda) are convenient for invitees who can add the meeting to their schedules and read the agendas.

Rationale: It is important to set a mutually agreed upon time and location for the initial meeting, especially because members of all three teams tend to be very busy. The location should either be colocated with the architects (which will help minimize disruptions for them) or be off-site so that requirements and architecture teams are less likely to be distracted or called away.

Preparation Task Postconditions

This task is successfully completed when the following postconditions are met:

- The assessment team has been properly staffed and trained in the assessment method.
- The top-level requirements and architecture teams have been identified and trained in the assessment method.
- The system architecture assessment initiation meeting has been organized.

5.1.2 System Architecture Assessment Initiation – Meeting

Meeting Task Objectives

The system architecture assessment phase meeting has the following major objectives:

1. Determine assessment scope.

A primary objective of this meeting is to determine the *scope* of the overall assessment including the types of architecturally significant (e.g., quality) requirements against which to assess the architectures of the system and its subsystems, the subsystems to be assessed, and the quantity of resources available to conduct the assessments:

- **Architecturally Significant Requirements**
  The importance of the different types and subtypes of architecturally significant requirements can vary greatly depending on the system whose architecture is being assessed. Typically, the architecturally significant requirements that are at highest risk for not being adequately supported by the architecture are the quality requirements. For example, performance is critical to real-time systems, interoperability is critical to systems that must communicate with many other systems, safety is obviously critical to safety-critical systems, and security can be critical to systems that must protect valuable assets from malicious attack.
- **Subsystems**
  The size of the system also impacts how many subsystems can be assessed given the limitations on project resources. If the system is decomposed into multiple subsystems, which are further decomposed into lower level subsystems, and so on, then the architecture is typically too large and complex to be assessed all at once. Often, multiple assessments are performed, one for each subsystem that is allocated important architecturally significant requirements. The subsystem architecture assessments are typically performed top-down by tier and horizontally within tiers based on the development schedule. Eventually, the subsystem assessment results are rolled up to produce the overall system assessment results.
- **Assessment Resources**
  The amount of time and effort available to perform the assessment also varies from system to system depending on the system's importance, schedule, and the availability of assessment and architecture team members.

2. Draft assessment schedule.

A second objective of the initiation meeting is to generate a general assessment schedule that enables the assessment team, the subsystem requirements team, and the architecture team to prepare for the subsystem assessments:

- **Schedule Sufficiency**
  The schedule of the requirements review meetings must allow the subsystem requirements teams to derive and the subsystem architecture teams to allocate the architecturally significant requirements to the subsystems being assessed.
- **Sufficient Yet Flexible Architecture**
  The schedule of the assessment meetings must allow the subsystem architecture teams sufficient time to create subsystem architectures that are sufficiently mature yet early enough to allow the architectures to be improved based on the results (i.e., the architectures cannot have been frozen and already used as the basis for significant design, implementation, and testing).[33]

[33] The need for the subsystem architecture that is being assessed to be sufficiently complete to support a meaningful assessment and yet flexible enough to be improved as a result of the assessment, can be difficult to achieve if an agile method is being used that promotes minimal, incremental, just-in-time architecture development as part of short duration iterations. In such methods where subsystem architecture, design, implementation, and test happen concurrently, any architectural problems identified may require abandonment of the associated design, implementation, and test.

- **Prior to Major Reviews**
  The subsystem architecture assessments are often scheduled prior to appropriate major project milestone reviews so that the results of the architectural assessments can be reported during the reviews.
- **Competing Schedule Pressure**
  Subsystem requirements reviews and subsystem architecture assessments compete with many other activities and tasks for limited resources. Specifically, assessors, requirements engineers, and architects are key personnel with very heavy schedules, making it difficult to develop assessment schedules when all stakeholders are available to prepare and take part.

3. Tailor architecture assessment method to be effective.

Unfortunately, conflicting goals of the assessment team and the subsystem requirements and architecture teams may make it difficult to achieve consensus on how to tailor the QUASAR method.

- **Assessment Team**
  The assessment team naturally wants to ensure an effective assessment and minimize the effort required to prepare for and perform the assessments. For example, they want an assessment method that provides them with highly relevant documentation organized by the type of architecturally significant requirement (e.g., quality factor). In contrast, the architecture team might feel it's sufficient to supply existing documentation, regardless of its organization or appropriateness as evidence.
- **Subsystem Requirements Teams**
  The subsystem requirements engineers naturally want to limit the impact of the requirements reviews on their requirements, many of which may already be frozen. They may not want to follow an assessment method that would generate significant, new, and verifiable quality requirements when they have concentrated heavily on functional requirements (e.g., by emphasizing use case modeling) and specified vague quality goals, rather than unambiguous quality requirements.
- **Subsystem Architecture Teams**
  The subsystem architects naturally want to limit the impact of the assessments on their effort because they are typically in the midst of developing and ensuring the integrity of their architectures. They may also not want to use an assessment method that causes them to develop significant new evidence for the assessments if they did not generate appropriate documentation during the normal course of developing their architectures.

Meeting Task Duration

The initial kickoff meeting can typically be completed in *one* day. However, the duration of the kickoff meeting depends on the ability of the assessment team and top-level architecture team to effectively collaborate to quickly achieve consensus on the scope of the architecture

---

Fortunately, there is a growing recognition of the need for a complete and stable architecture prior to the agile development of the associated design, implementation, and test work products.

assessments in terms of the types of architecturally significant requirements that will form the basis of the assessments, the subsystems to assess, the assessment schedule, and the details of how to tailor the assessment method.

Meeting Task Preconditions
This task can be started when the following preconditions are met:

- The assessment team has been properly staffed and trained in the assessment method.
- The top-level requirements and architecture teams have been identified and trained in the assessment method.
- The system architecture assessment initiation meeting has been organized.

Meeting Task Steps
During the initial kickoff meeting, complete the following steps:

1. Set scope.

Step: The assessment team and the top-level requirements and architecture teams collaborate to build a consensus on the scope of the assessments.[34]

Rationale:

- It is critical to get an up-front consensus among the major stakeholders on the scope of the overall assessment.
- There should be (but often is not) a general agreement among the three teams as to what are the most important quality factors against which to assess the system architecture. However, depending on the application domain, size, and complexity of the system, quality factors that are important for some subsystems may not be important for others. It is therefore important to remember that this is merely an initial list, which may be later modified to meet the assessment needs of individual subsystem architectures.

a. *Subsystems*

Substep: Take a risk-based approach to identify the subsystems, the architectures of which are to be assessed.
Rationale: Sufficient resources are probably not available to assess all subsystems. Limited assessment resources should be invested in assessing the architectures of

- mission- or safety-critical subsystems
- complex and/or large subsystems
- subsystems using previously untested architectures or technologies
- subsystems that are similar to subsystems that have previously had problems due to poor architectures

[34] Note that this implies that the assessment team has the knowledge and authority to speak for the acquisition organization and that the top-level requirements and architecture teams have the knowledge and authority to "speak" for the development organization with regard to establishing the scope of the assessments.

- subsystems whose architectures are being produced by relatively inexperienced architects
- subsystems that need to be certified (e.g., safety certifications, security certifications); the assessment ensures that arguments and evidence that will help convince the certifiers are developed

b. *Quality Factors*

Substep: Based on the subsystems selected to be assessed, create an initial prioritized list of the important types of architecturally significant quality requirements against which the subsystems will be assessed.[35]
Rationale: Not all quality factors are relevant and the relevant quality factors are not equally important. There is typically not sufficient time to assess the subsystem architecture's support for all relevant quality factors, so resources must be invested in assessing those quality factors having the highest associated risk and largest benefit.

c. *Resources*

Substep: Set the resources per method task in terms of personnel and schedule (e.g., number of days to allocate to preparation, meeting, and follow-through).
Rationale: Limited resources must be applied wisely.

2. Develop schedule.

Step: The assessment team and the top-level requirements and architecture teams collaborate to develop an initial version of architecture assessment schedule. Although this schedule should be relatively precise for the dates of the requirements reviews and architecture assessments for those subsystems to be assessed first, the schedule may be fairly vague about the dates of later reviews and assessments.

Rationale: A rough overall schedule is needed early to ensure that resources can be available when needed and that the assessment tasks do not conflict with other important project tasks.

3. Tailor assessment method.

Step: If necessary, the assessment team and the top-level requirements and architecture teams collaborate[36] to tailor the QUASAR method to better meet the specific needs of the system or subsystem being assessed. Tailoring typically involves modifying the method's

[35] Note that would be very difficult without the top-level requirements team, which should be eliciting this kind of information from the system's stakeholders, and the top-level architecture team, which should be able to help prioritize the resulting quality factors in terms of how they will impact the system architecture.

[36] To enable the method to meet the specific characteristics of individual subsystem architectures, further tailoring may also need to be made on an individual subsystem-by-subsystem basis. However, care must be taken to avoid unnecessary subsystem-level tailoring in order to ensure an adequate assessment of each selected subsystem's architecture and to enable the assessment team to obtain consistent and comparable assessment results.

- *Tasks*
  The tasks performed during the different phases of the assessment can be tailored. New tasks can be added, existing tasks can be deleted, and steps composing the tasks can be added, deleted, or modified. Task durations can also be modified to fit the schedule and resource availability.
- *Teams*
  The teams performing these tasks can be tailored by adding new teams, combining existing teams, deleting teams deemed unnecessary or unavailable, and modifying existing teams' responsibilities or membership.
- *Work Products*
  The work products produced during these tasks and steps can be added, modified, or deleted. Typical tailoring includes modifying the stakeholders for the assessments or contents of documents.

Rationale: It is important that the assessment team and both the top-level requirements and architecture teams agree on the architecture assessment method so that it benefits all three teams. The method may need to be tailored to meet the specific needs of the project or subsystem being assessed in order to be

- effective in terms of developing a correct assessment
- efficient in terms of effort required to perform the assessment
- feasible in terms of necessary staffing availability, funding, and schedule.

4. Manage action items.

Step: The assessment team collaborates with the top-level requirements and architecture teams to collect, identify, and record any action items from the meeting. This includes setting due dates and assigning action items to appropriate people. Examples of common action items include the following:

- obtain any delayed input on assessment scope (e.g., availability of resources and inclusion of subsystems and related types of quality requirements)
- approve pending subsystem assessment schedules
- gather final input on assessment method tailoring from development organization process engineering team

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and tracked to completion.

Meeting Task Postconditions

This task is successfully completed when the following postconditions are met:

- The assessment scope has been determined.
- The initial assessment schedule has been set.
- The system architecture quality assessment has been tailored and agreed upon by the attendees.

5.1.3 System Architecture Assessment Initiation – Follow-Through

Follow-Through Task Objectives
The follow-through task of the system architecture assessment initiation phase has the following major objectives:

- Report meeting results. The results of the initial kickoff meeting are reported to stakeholders.
- Ensure adequate support. By distributing the schedule and clarifying the responsibilities and tasks of the various teams involved, the follow-through task helps to ensure that the requirements reviews and architecture assessments are officially scheduled and resources.
- Track action items. Action items identified during the system architecture assessment initiation meeting are tracked to closure.

Follow-Through Task Duration
The duration of this task largely depends on the availability of the members of the assessment and top-level architecture teams and can last anywhere from a few days to two weeks. Naturally, completing this task should be a high priority so that the outputs can be distributed while they remain fresh in everyone's minds.

Meeting Task Preconditions
This task can be started when the following preconditions are met:

- The assessment scope has been determined.
- The initial assessment schedule has been set.
- The system architecture quality assessment has been tailored and agreed upon by the stakeholders.

Follow-Through Task Steps
After the initial assessment kickoff meeting but prior to the first assessment, the following steps are performed in a timely manner:

1. **Produce, review, and present system architecture assessment initiation meeting outbrief.**

   Step: Within a day or two of the initiation meeting and usually before leaving the site of the meeting, complete the following steps:

   a. The members of the assessment team (especially the scribe) provide the leader of the assessment team with the team members' meeting notes.
   b. The leader of the assessment team produces an initial version of the initiation meeting outbrief.
   c. The assessment team members review the outbrief to ensure that it correctly summarizes the results of the initiation meeting.

d. The leader of the assessment team iterates the outbrief to incorporate the comments and recommendations of the assessment team members.
e. The leader of the assessment team presents the outbrief to the attendees and available stakeholders of the initiation meeting.

Rationale: It is important to provide an informal outbrief at the end of the system architecture assessment initiation meeting so that attendees and stakeholders do not wait several weeks to obtain a clear overview of the results of the meeting.

2. Produce, review, and distribute system architecture assessment initiation meeting minutes.

Step: Within a couple of weeks of the system architecture assessment initiation meeting, complete the following steps:

a. Members of the assessment team provide their notes to the members of the team tasked to produce the initiation meeting minutes.[37]
b. The members of the assessment team so tasked produce an initial version of the meeting minutes.
c. The assessment team members review the meeting minutes to ensure that they correctly record the results of the initiation meeting.
d. The assessment team leader iterates the meeting minutes to incorporate the comments and recommendations of the assessment team members.
e. The leader of the assessment team leader distributes the meeting minutes to stakeholders, especially the meeting attendees.

Rationale: The consensus concerning the scope of the assessment developed during the meeting needs to be officially documented to minimize the chance of confusion or potential future repudiation by any attendees.

3. Tailor architecture assessment procedure.

Step: Within a couple of weeks of the system architecture assessment initiation meeting, complete the following steps:

a. Using the initial kickoff meeting notes and other sources, assigned members of the assessment team update the architecture assessment procedure and distribute the resulting tailored version to members of the assessment team.
b. Members of the assessment team respond with comments and recommended changes.
c. After iterating the architecture assessment procedure, the assessment team distributes the updated version of the architecture assessment procedure to the top-level requirements and architecture teams, which forward the procedure to the other development stakeholders such as the leads (or members) of the subsystem requirements and architecture teams.

---

[37] A subset of the assessment team may be tasked to produce the entire requirement review meeting minutes. On the other hand, the assessment team may be divided into pairs that are tasked to produce different subsections of the meeting minutes. The choice of approach may vary depending on the size and formality of the requirements review meeting minutes, which could contain a significant number of observations and recommendations. In fact, depending on the importance given to the requirement review, a report may replace less formal meeting minutes.

d. The subsystem requirements and architecture teams respond with comments and recommendations to the top-level requirements and architecture teams.
e. After organizing (and potentially passing judgment on) these comments and recommendations, the top-level requirements and architecture teams pass the approved and organized comments and recommendations back to the assessment team.
f. The assessment team iterates the procedure.
g. Upon approval by the leader of the assessment team, the project-specific system architecture assessment procedure is distributed to all stakeholders.

Rationale: Stakeholders (and especially participants) need to understand the method by which the architectures will be assessed. They also need to be able to submit recommended modifications for project-specific tailoring. However, the assessment method ultimately belongs to the assessment team, which determines the official project-specific version of the method used.

4. Distribute schedule.

Step: The assessment team, the top-level requirements team, and top-level architecture team distribute the assessment schedule to all stakeholders including managers and the subsystem requirements and architecture teams.

Rationale: Stakeholders need to know the tentative schedule for the various subsystem assessments so that they can plan other activities accordingly. Stakeholders also need to ensure that the subsystem architecture quality assessments are scheduled to support major system reviews such as PDRs.

5. Obtain needed resources.

Step: Estimate the resources needed to perform the assessments, officially fund and schedule the initial assessments, and assign staff.

Rationale: Without proper funding, scheduling, and staffing, necessary resources will not be available to perform the assessments, resulting in ineffectual assessments that may not be held in time to effectively influence the system and subsystem architectures.

6. Track action items to completion.

Step: The assessment team tracks open action items to completion.

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and tracked to completion.

7. Capture lessons learned.

Step: The assessment team collaborates with the system architecture team to capture lessons learned about the effectiveness of the system architecture assessment initiation phase of the QUASAR method.

Rationale: This enables the method to be continually improved and tailored for future system architecture assessment initiation meetings and for future projects.

8. **Update assessment method and associated training materials.**

   Step: Based on the lessons learned, the project process team[38] updates the project-tailored QUASAR architecture assessment procedure and the associated QUASAR training materials. Where appropriate, they also provide the SEI with the lessons learned and updated procedure and training materials so that the official QUASAR documentation can be iterated.

   Rationale: Updating the QUASAR method will enable the process to be improved for future projects.

5.2 Subsystem Requirements Review Phase

Phase Objectives
The objectives of this phase is to ensure that the

- architecturally relevant quality goals and requirements are properly derived and allocated to each subsystem for which the architecture is to be assessed
- subsystem architecture team understands what is expected from them during the subsystem architecture assessment phase

Phase Duration
Whereas the subsystem requirements review meeting may only last one or two days, preparations may begin up to one month before the meeting; follow-through may last a few weeks after the meeting.

Phase Tasks
During this phase, perform the following tasks for each subsystem (or other major element of the architecture) to be assessed:

1. **Subsystem Requirements Review – Preparation**

   The assessment team and subsystem architecture team prepare by exchanging and reading relevant preparatory documentation.

2. **Subsystem Requirements Review – Meeting**

   The subsystem architecture team demonstrates to the assessment team their knowledge of the architecturally significant requirements and their understanding of what is expected of them during the coming assessment.

3. **Subsystem Requirements Review – Follow-Through**

   The assessment team reports their findings and recommendations to the subsystem architecture team and other stakeholders.

[38] If an official “project process team” (e.g., system engineering process group [SEPG]) does not exist, then whichever team, group, or individual that is filling the process engineer role should perform this task.

Figure 14 illustrates the decomposition of Subsystem Requirements Review Phase into its three tasks, their associated work products, and the associated teams that produce, use, and update these work products.

5.2.1 Subsystem Requirements Review – Preparation

Preparation Task Objective
The objective of this task is to ensure that the members of the assessment team, subsystem requirements team, and subsystem architecture team are properly prepared for the requirements review meeting.

Preparation Task Duration
The duration of this task largely depends on the availability of the members of the assessment team, subsystem requirements team, and subsystem architecture team. This task typically lasts up to one month. Although it takes time to produce and properly review the preparatory materials, members of the teams should strive for the shortest practical duration so as to maintain continuity and establish a proper sense of importance and urgency.

Preparation Task Preconditions
This task can be started when the quality requirements have been derived and allocated to a subsystem, the architecture of which is to be assessed.

Preparation Task Steps
Prior to the subsystem requirements meeting, perform the following steps:

1. **Train subsystem teams.**

   Step: Up to approximately one month prior to the requirements meeting, provide the assessment team members of the subsystem requirements and subsystem architecture teams with training in the tailored QUASAR system method so that they can properly prepare for the meeting. As part of this training, the assessment team provides each member of the subsystem requirements and subsystem architecture teams with a copy of the architecture assessment procedure, the architecture assessment training materials, and the subsystem requirements review.

   Rationale: The members of the subsystem requirements and subsystem architecture teams need to understand the method in which they will take part, especially their responsibilities including the steps they will perform and the work products they will need to produce. Members of these two teams need to find adequate time to learn the method.

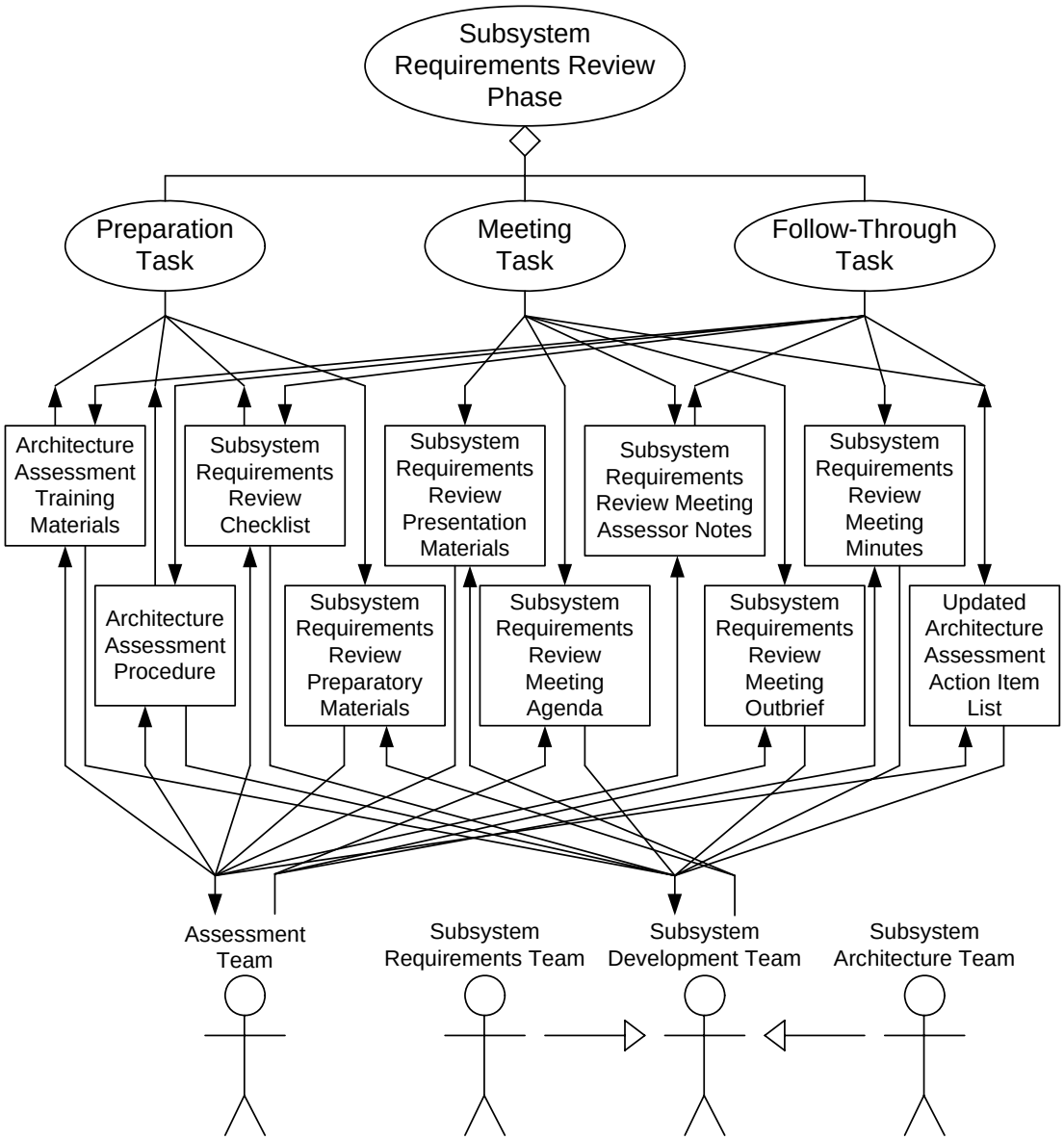

Figure 14: Subsystem Requirements Review Phase

2. Provide a requirements review checklist.

Step: The assessment team supplies the subsystem requirements and subsystem architecture teams with the requirement review checklist that the assessment team will use during the requirements review meeting.

Rationale: This will enable the subsystem requirements and subsystem architecture teams to better prepare for the coming assessment.

3. Internally review quality goals and requirements.

Step: Approximately one month prior to the requirements meeting, the subsystem architecture team collaborates with the subsystem requirements team to ensure that all architecturally significant quality goals and requirements are properly derived and allocated

(i.e., traced) to their subsystems. The teams ensure that the architecturally significant requirements are prioritized and scheduled for implementation based on criteria such as:

- criticality to users and the acquisition organization
- architectural and implementation difficulty and risk
- proper implementation order (For example, certain quality factors [e.g., security] need to be built into the system early because they would require major changes to the architecture if incorporated during later builds.)

Rationale: It is critical for the subsystem architecture team to understand the architecturally significant goals and requirements that will drive their architectures.

4. **Develop and provide preparatory materials.**

   Step: Approximately three weeks prior to the requirements meeting, the subsystem requirements and subsystem architecture teams develop the subsystem requirements review preparatory materials. The subsystem requirements team collects (e.g., either physically or via meta tags) and organizes these architecturally significant quality goals and requirements to produce their part of the preparatory materials. The subsystem architecture team generates a partial, representative architecture quality case including claims, arguments, and evidence. Both teams provide the preparatory material to the assessment team. The subsystem architecture team also provides the assessment team with *read-only* access to the relevant derived and allocated architecturally significant quality goals and requirements in its requirements repository.

   Rationale: By providing the assessment team with the subsystem requirements review preparatory materials at least three weeks prior to the review, the assessment team has sufficient time to familiarize themselves with the materials. This will make the assessment team more effective and efficient at the actual review meeting, thereby enabling the meeting to be shorter in duration. In turn, this will enable the members of the subsystem requirements and subsystem architecture teams to spend less time away from their primary work of engineering requirements and architecting the subsystem.

5. **Develop and provide presentation materials.**

   Step: During the weeks immediately before the requirements meeting, the subsystem requirements and subsystem architecture teams develop the presentation materials and provide them to the members of the assessment team.

   Rationale: The rationale for providing early access to the subsystem requirements review presentation materials is similar to that for the preparatory materials.

6. **Become familiar with the subsystem.**

   Step: The assessment team properly prepares for the requirements review meeting by reading the preparatory and presentation materials provided by the subsystem requirements and architecture teams. The assessment team familiarizes themselves with the subsystem and its architecturally significant quality goals and requirements. They assessment team also familiarizes themselves with the architectural quality cases that the

subsystem architecture team intends to present during the subsystem architecture assessment phase.

Rationale: By familiarizing themselves with the subsystem, its relevant quality goals and requirements, and the maturity of the architecture team's intended quality cases prior to the requirements meeting, the assessment team can be more efficient and effective during the meeting.

7. **Organize the meeting.**

   Step: At least two weeks prior to the subsystem requirements review meeting, the assessment team, subsystem requirements team, and subsystem architecture team collaborate to complete the following tasks:

   a. **Identify meeting attendees and other stakeholders.**

      Substep: The assessment, subsystem requirements, and subsystem architecture teams collaborate to develop a list of meeting attendees and other stakeholders who should receive copies of the subsystem requirements review meeting preparatory and presentation materials.

      Rationale: It is important to ensure that no significant stakeholder is overlooked.

   b. **Set time and location.**

      Substep: The assessment, subsystem requirements, and subsystem architecture teams collaborate to set an exact time and location (e.g., address, building number, and room number) for the subsystem requirements review meeting.

      Rationale: Attendees need an exact time and place to avoid schedule conflicts and make necessary arrangements.

   c. **Develop meeting agenda.**

      Substep: The assessment, subsystem requirements, and subsystem architecture teams collaborate to produce a meeting agenda covering the relevant quality factors, associated quality goals and requirements, sample quality cases, and meeting wrap-up (e.g., status of action items and assignment of final new action items).

      Rationale: Agendas enable attendees to identify the most important parts of the meeting.

   d. **Invite stakeholders.**

      Substep: The assessment team sends invitations including the meeting agenda to the identified meeting attendees and other stakeholders.

      Rationale: Documented invitations (e.g., email with attached agenda) are convenient for invitees who can add the meeting to their schedules and read the agendas.

Preparation Task Postconditions

This task is successfully completed when the following postconditions are met:

- The subsystem requirements and architecture teams have been trained in the assessment method.

- The subsystem quality requirements have been derived, prioritized, and allocated to the subsystem.
- The subsystem requirements review preparatory and presentation materials have been developed and provided to the assessment team.
- The assessment team has become familiar with the subsystem and the associated preparatory and presentation materials.
- The subsystem requirements review meeting has been organized.

5.2.2 Subsystem Requirements Review – Meeting

Meeting Task Objectives
The objectives of this task are to ensure that the

- quality requirements have been properly derived and allocated to the subsystem
- assessment team understands what is expected of them during the following Subsystem Architecture Assessment Phase

Meeting Task Duration
This task usually only lasts one day per subsystem, but it may vary depending on the number of subsystem quality requirements and the preparedness of the subsystem architecture team.

Meeting Task Preconditions
This task can be started when the following preconditions are met:

- The subsystem requirements and architecture teams have been trained in the assessment method.
- The subsystem quality requirements have been derived, prioritized, and allocated to the subsystem.
- The subsystem requirements preparatory and presentation materials have been developed and provided to the assessment team.
- The assessment team has become familiar with the subsystem and the associated preparatory and presentation materials.
- The subsystem requirements review meeting has been organized.
- Action items have been identified, assigned, and scheduled.

Meeting Task Steps
Perform the following steps during each requirements review meeting:

1. **Present quality factors.**

   Step: The subsystem architecture team demonstrates to the assessment team their understanding of the architecturally significant quality factors.

Rationale: Misunderstandings concerning the definition and relative importance of the quality factors is not uncommon. It is critical to forge a consensus between the requirements team, architecture team, and assessment team as to the architecturally significant quality factors, subfactors, and their meanings.

2. **Present quality requirements.**

   Step: The subsystem requirements team presents a summary of the architecturally significant quality requirements to the assessment team. The subsystem requirements team also presents their prioritization and scheduling of implementation of these requirements. The assessment team raises any concerns they might have that this prioritization and scheduling of requirements may not

   – be consistent with the needs of the users and acquisition organization
   – minimize architectural risks

   Rationale: This is a good time to ensure that the architecture team understands the architecturally significant quality requirements. This step also gives the assessment team confidence that the architects will incorporate these requirements into the subsystem architecture. It is critical that architectural implementation be properly scheduled in terms of

   – criticality to users and the acquisition organization
   – architecture and implementation difficulty and risk
   – proper implementation order (For example, certain quality factors [e.g., security] need to be built into the system early because they would require major changes to the architecture if incorporated during later builds.)

3. **Present sample quality case information.**

   Step: The subsystem architecture team presents a small representative sample of the kind of quality case information they intend to present during the following Subsystem Architecture Assessment Phase. They demonstrate to the assessment team that they understand what is expected of them during the coming assessment meeting by giving a few brief representative examples of their intended arguments that their architecture will meet its allocated quality-related requirements (i.e., list some important architectural decisions that support these requirements) and associated evidence (e.g., official documentation of these architectural decisions).

   Rationale: Misunderstandings concerning the proper contents of quality cases are not uncommon. This step gives the assessment team confidence that the architects will provide proper quality cases during the following architecture assessment phase.

4. **Provide guidance.**

   Step: The assessment team gives the subsystem requirements team recommendations for improving the quality (e.g., completeness, lack of ambiguity, and verifiability) of the quality requirements. The assessment team gives the architecture team recommendations for improving their proposed quality cases (e.g., arguments and evidence). For example, they should avoid using plans and development processes as “arguments” and planning

and procedures documents as “evidence.” These documents are inappropriate because they are not architectural decisions and not documentation of architectural decisions. The assessment team also answers any questions that members of the subsystem requirements and architecture teams might have (e.g., regarding quality cases and the coming architecture assessment meeting).

Rationale: In spite of training provided during the preceding subsystem requirements review preparation task, members of the requirements team can often benefit from guidance regarding the proper form of quality requirements, which will enable them to improve the requirements before the architecture team creates the subsystem architecture. Similarly, the architecture team’s need for guidance in the proper form and content of quality cases is often only clear once they present their initial representative examples of quality cases.

5. **Determine special assessment team staffing needs.**

   Step: The assessment team and subsystem architecture team collaborate to determine if the assessment team needs to be augmented with members having particular expertise such as specialty engineering expertise (e.g., reliability, safety, or security engineering) or expertise in important application domains (e.g., communications or sensor technology).

   Rationale: It is important that the assessment team include members with adequate training and experience to properly assess the architecture of the subsystem, including people with both specialty engineering or application domain expertise. The subsystem architecture team can best identify any important application domains that apply to their subsystem.

6. **Schedule coming events.**

   Step: The assessment team and subsystem architecture team collaborate to schedule the coming assessment meeting and its associated preparation steps (e.g., dates by which the architecture team will provide the assessment team with advanced access to their arguments and evidence).

   Rationale: Scheduling enables stakeholders to update their schedules and avoid schedule conflicts.

7. **Manage action items.**

   Step: The assessment team collaborates with the subsystem requirements and subsystem architecture teams to collect, identify, and record any action items from the meeting. This includes setting due dates and assigning the action items to appropriate people. Examples of common action items include

   – Members of the subsystem requirements team and subsystem architecture team are assigned requests for information (RFI) and requests for action (RFA) by the assessment team.

- The assessment team members are assigned the tasks of supplying their requirements review meeting notes to the team scribe.
- The assessment team leader is assigned the tasks of producing the subsystem requirements review outbrief.
- The assessment team members are assigned the task of reviewing the outbrief before it is presented to the subsystem requirements team, subsystem architecture team, and any other meeting attendees or stakeholders (e.g., managers and members of the top-level architecture team).
- The assessment team leader presents the outbrief to the meeting attendees and any other interested stakeholders.
- Assessment team members are assigned the tasks of producing, reviewing, and distributing the subsystem requirements review meeting minutes.

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and properly tracked to completion.

Meeting Task Postconditions
This task is successfully completed when the following postconditions are met:

- The subsystem requirements team has presented the subsystem quality factors and associated subsystem quality requirements.
- The subsystem architecture team has presented sample subsystem quality cases.
- The assessment team has provided guidance to the subsystem requirements and architecture teams.
- The meeting attendees have collaborated to determine if any special assessment staffing needs exist.
- The subsystem architecture assessment meeting has been scheduled.

5.2.3 Subsystem Requirements Review – Follow-Through

Follow-Through Task Objective
The objectives of this task are to ensure that the

- requirements review outbrief is produced and presented to the relevant stakeholders
- meeting minutes are produced and distributed to relevant stakeholders
- action items are tracked to completion

Follow-Through Task Duration
This task usually takes

- several days, depending on staffing availability, to complete the meeting minutes
- a few weeks to track action items to completion, depending on the number and type of action items

Follow-Through Task Preconditions

This task can be started when the following preconditions are met:

- The subsystem requirements team has presented the subsystem quality factors and associated subsystem quality requirements.
- The subsystem architecture team has presented sample subsystem quality cases.
- The assessment team has provided guidance to the subsystem requirements and architecture teams.
- The meeting attendees have collaborated to determine if any special assessment staffing needs exist.
- The subsystem architecture assessment meeting has been scheduled.

Follow-Through Task Steps

After each requirements review meeting but before the associated architecture assessment meeting, the following steps are performed in a timely manner:

1. **Produce, review, and present requirements review outbrief.**

   Step: Within a day or two of the requirements review meeting and before leaving the site of the requirements review

   a. The members of the assessment team (especially the assessment team scribe) provide the leader of the assessment team with their meeting notes.
   b. The leader of the assessment team produces an initial version of the requirement review outbrief.
   c. The assessment team members review the outbrief to ensure that it correctly summarizes the results of the subsystem requirements review.
   d. The leader of the assessment team iterates the requirements review outbrief to incorporate the comments and recommendations of the assessment team members.
   e. The leader of the assessment team presents the outbrief to the attendees and available stakeholders of the subsystem requirements review.

   Rationale: It is important to provide an informal outbrief at the end of the requirements review meeting so that attendees and stakeholders need not wait several weeks to obtain a clear indication of the results of the meeting.

2. **Produce, review, and distribute requirements review meeting minutes.**

   Step: Within a couple of weeks of the requirements review meeting

   a. Members of the assessment team provide their notes to the members of the team tasked to produce the requirements review meeting minutes.[39]

[39] A subset of the assessment team may be tasked to produce the entire requirement review meeting minutes. On the other hand, the assessment team may be divided into pairs which are tasked to produce different subsections of the meeting minutes. The choice of approach may vary depending on the size and formality of the requirements review meeting minutes, which could contain a significant number of observations and recommendations. In fact, depending on the importance given to the requirement review, a report may replace less formal meeting minutes.

b. The selected members of the assessment team produce an initial version of the requirements review meeting minutes.
c. The assessment team members review the requirements review meeting minutes to ensure that they correctly record the results of the subsystem requirements review.
d. The assessment team leader iterates the requirements review meeting minutes to incorporate the comments and recommendations of the assessment team members.
e. The leader of the assessment team leader distributes the requirements review meeting minutes to stakeholders.

Rationale: It is important to document relevant observations and agreements (e.g., schedule dates) in a form available to all relevant stakeholders.

3. Track action items to completion.

Step: The assessment team tracks open action items to completion.

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and properly tracked to completion.

4. Capture lessons learned.

Step: The assessment team collaborates with the subsystem requirements and subsystem architecture teams to capture lessons learned about the effectiveness of the subsystem requirements review phase of the QUASAR assessment.

Rationale: This enables the method to be continually improved and tailored for future subsystem requirements reviews and use on future projects.

5. Update assessment method and associated training materials.

Step: Based on the lessons learned, the project process team[40] updates the project-tailored QUASAR assessment procedure and the associated QUASAR training materials. Where appropriate, they also provide the SEI with the lessons learned and updated procedure and training materials so that the official QUASAR documentation can be iterated.

Rationale: Updating the QUASAR enables the process to be improved on future subsystem requirements reviews and other projects.

Follow-Through Task Postconditions

This task is successfully completed when the following postconditions are met:

- The subsystem requirements team has presented the subsystem quality factors and associated subsystem quality requirements.
- The subsystem architecture team has presented sample subsystem quality cases.

[40] If an official “project process team” (e.g., system engineering process group [SEPG]) does not exist, then whichever team, group, or individual that is filling the process engineer role should perform this task.

- The assessment team has provided guidance to the subsystem requirements and architecture teams.
- The meeting attendees have collaborated to determine if any special assessment staffing needs exist.
- The subsystem architecture assessment meeting has been scheduled.
- Open action items have been tracked to completion.

5.3 Subsystem Architecture Assessment Phase

Phase Objective
The objective of this phase is to assess the quality of a subsystem architecture in terms of its derived and allocated architecturally significant requirements.

Phase Duration
The duration of this phase typically lasts a few weeks, with most of the time being spent on preparation and follow-through, rather than on the actual assessment meeting, which should last only one or two days.

Phase Tasks
As illustrated in Figure 15, perform the following tasks for each subsystem (or other major element of the architecture) to be assessed:

1. **Subsystem Architecture Assessment – Preparation**

   The assessment team prepares by exchanging and reading relevant documentation provided by the subsystem architecture team.

2. **Subsystem Architecture Assessment – Meeting**

   The subsystem architecture team presents to the assessment team quality cases showing that their architecture adequately supports its relevant architecturally significant requirements.

3. **Subsystem Architecture Assessment – Follow-Through**

   The assessment team reports their findings and recommendations to the subsystem architecture team and other stakeholders.

5.3.1 Subsystem Architecture Assessment – Preparation

Meeting Task Objectives
The objectives of this task are as follows:

- The architecture team presents their quality cases to the assessment team.
- The assessment team actively questions the architecture team to identify

– architecture defects and weaknesses (both by commission and omission)
– architectural risks

Preparation Task Duration

The duration of this task is typically a few weeks per subsystem. Note that preparation does *not* include developing the subsystem architecture, which should naturally include the development of the architects' arguments (architectural decisions including rationales) and result in documentation that will become the quality case evidence. That being said, the duration may still vary depending on the

- size and complexity of the subsystem architecture
- number of quality cases to be developed and presented
- amount of preparatory materials to be produced and read
- skill, expertise, and productivity of the teams involved

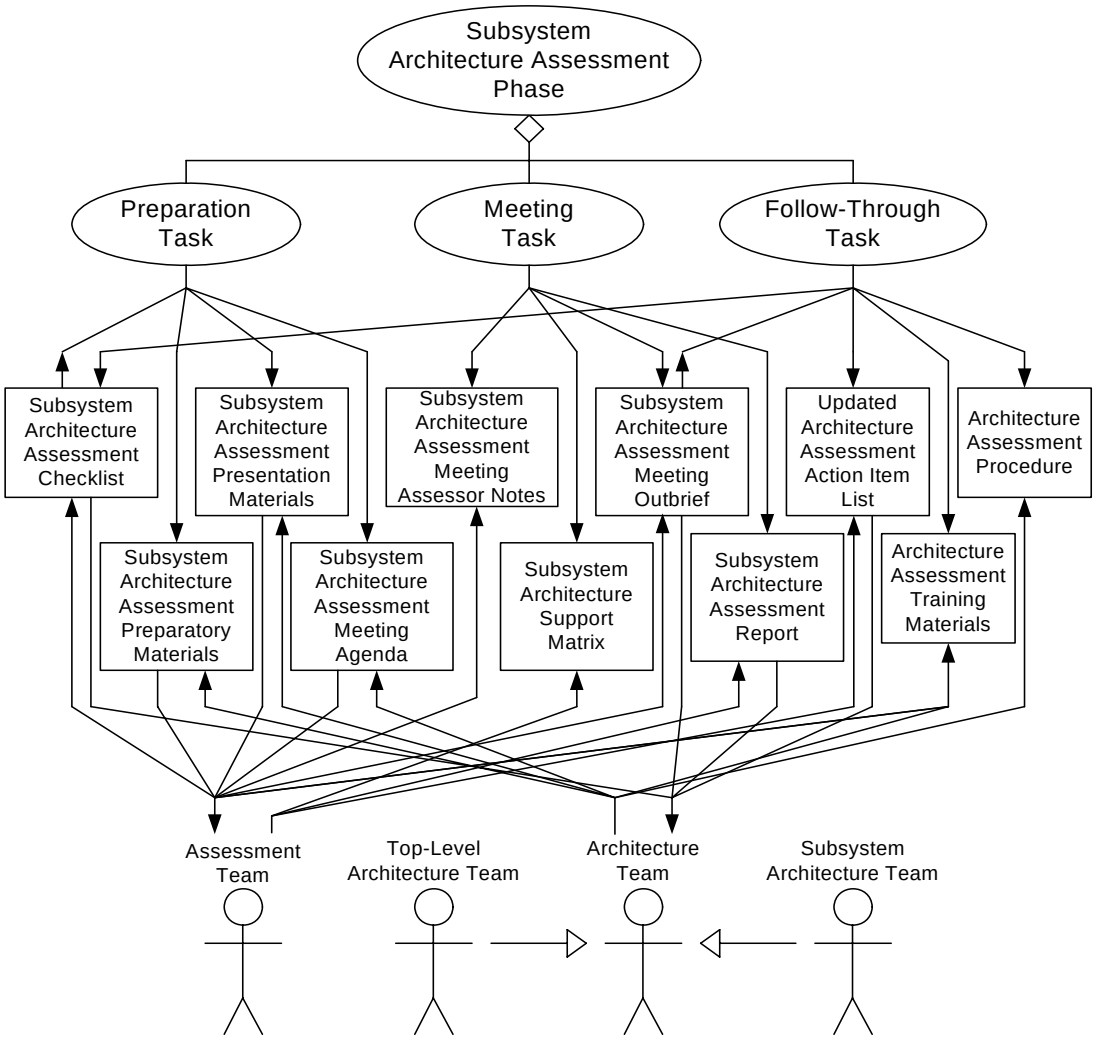

Figure 15: Subsystem Architecture Assessment Phase

Preparation Task Preconditions

This task can be started when the subsystem architecture is sufficiently complete and mature enough to be assessed.

Preparation Task Steps

Prior to each subsystem architecture assessment meeting, perform the following steps:

1. **Provide architecture assessment checklist.**

   Step: The assessment team supplies the subsystem architecture team with the architecture assessment checklist that they will use during the assessment.

   Rationale: This enables the subsystem architecture team to better prepare for the coming assessment.

2. **Generate preparatory materials including architecture quality cases.**

   Step: The subsystem architecture team develops the subsystem architecture assessment preparatory materials.

   Rationale: These materials help the assessment team better prepare for the assessment meeting.

   a. **Gather subsystem architecture overview.**

      Substep: The subsystem architecture team gathers (or generates if necessary) an overview of the subsystem architecture (e.g., architecture training material).

      Rationale: An architecture overview helps the assessment team better understand the subsystem architecture team's quality cases.

   b. **Gather updated quality requirements.**

      Substep: The subsystem architecture team gathers any updates to the architecturally significant requirements that are derived and allocated to the subsystem.

      Rationale: Such requirements drive the architecture and associated quality cases. The assessment team may not know about changes that have occurred since the subsystem requirements review.

   c. **Gather quality cases.**

      Substep: The subsystem architecture team gathers (or generates if necessary[41]) the architecture quality cases for each appropriate quality factor and quality subfactor.

      Rationale: The most important part of the subsystem architecture assessment preparatory materials are the quality cases. The assessment team needs to have these

[41] Optimally, the architects should incrementally develop and document their architecture quality cases as they architect subsystems. Specifically, arguments and associated evidence should be created as a natural part of the architectural documentation. Unfortunately, most architectural documentation does not adequately document the decisions that the architects made to achieve adequate quality and the associated rationales. Even if the information exists, it is often scattered throughout large amounts of other architectural information, making it difficult for the assessors to access.

early in order to prepare for the meeting (e.g., develop questions for the subsystem architecture team).

- *Gather claims.*
  Substep: Based on the relevant quality factors and subfactors, the subsystem architecture team gathers (or generates) claims that their architecture adequately supports achieving its allocated quality goals and meets its associated allocated and derived quality-related requirements.
  Rationale: The assessment team needs to clearly understand the architects' claims if they are to decide if the arguments and supporting evidence justify belief in the claims. In other words, the assessors need to know how good the architecture has to be.
- *Generate arguments.*
  Substep: The subsystem architecture team generates clear and compelling arguments as to why their architecture adequately supports its allocated and derived quality-related requirements.
  Rationale: The arguments are the key to the quality case, informing the assessors of the architectural decisions and their rationales. The assessment team cannot adequately assess the quality of the architecture without hearing these arguments.
- *Gather supporting evidence.*
  Substep: The subsystem architecture team collects and organizes official evidence supporting their arguments. They provide an index, pointers, or some other means to identify the relevant information.
  Rationale: The assessment team needs to sample a representative collection of the evidence to ensure that the architecture decisions documented in the arguments actually exist. These evidentiary diagrams, models, and documents provide insight into the architecture beyond the arguments, enabling the assessment team to understand the strengths and weaknesses of the architecture's support for the various types of quality requirements.

3. Develop subsystem architecture presentation materials.

Step: The subsystem architecture team develops the materials that they are going to present to the assessment team during the subsystem architecture assessment meeting.

Rationale: The presentation materials are provided to the assessment team prior to the assessment to help them prepare, making the actual assessment meeting more effective and productive.

a. Create a brief subsystem architecture overview.

Substep: The subsystem architecture team generates a very brief presentation overview of the subsystem architecture. It summarizes the subsystem introduction contained in the subsystem preparation materials.

Rationale: During the subsystem architecture assessment meeting, a very brief architecture overview helps the assessment team understand the subsystem architecture team's quality cases. This is especially helpful for attendees who did not read the preparatory materials.

b. Create a brief summary of the architecture quality cases.

Substep: On a quality factor-by-quality factor basis, the subsystem architecture team generates a brief summary presentation of each architecture quality case.

Rationale: Due to time constraints, the subsystem architecture team presents a brief summary of the architecture quality cases to the assessment team during the subsystem architecture assessment meeting. The subsystem architecture team also provides the assessment team access to these summary quality cases ahead of the meeting, so that the meeting is effective and productive.

4. **Make preparatory and presentation materials available.**

   Step: The subsystem architecture team provides the assessment team with electronic access to their architecture assessment preparatory and presentation materials the agreed upon number of weeks (typically two to three) prior to the architecture assessment meeting.

   Rationale: The members of the assessment team require adequate lead time to read the preparatory and presentation meetings materials.

5. **Obtain and read preparatory and presentation materials.**

   Step: Prior to the subsystem architecture assessment meeting, the assessment team obtains and reads the architecture team's preparatory and presentation materials.

   Rationale: This enables the assessment team to properly prepare for the assessment.

6. **Submit preliminary RFIs and RFAs.**

   Step: Prior to the subsystem architecture assessment meeting, and based on their review of the preparatory and presentation materials, members of the assessment team potentially develop and submit to the subsystem architecture team RFIs and RFAs. RFIs usually consist of questions and requests for clarifications of the architecture quality cases and RFAs are requests for additional evidence or pointers to relevant parts of large and complex evidentiary documentation.

   Rationale: This enables the subsystem architecture team to supply the requested information either prior to or during the subsystem architecture assessment meeting.

7. **Organize the meeting.**

   Step: At least two weeks prior to the subsystem architecture assessment meeting, the assessment team and subsystem architecture team collaborate to

   a. **Identify meeting attendees and other stakeholders.**

      Substep: The assessment team and subsystem architecture team develop a list of meeting attendees and other stakeholders who should receive copies of the meeting preparatory and presentation materials.

      Rationale: It is important to ensure that no significant stakeholder is overlooked.

   b. **Set time and location.**

      Substep: The assessment team and subsystem architecture team determine an exact time and location (e.g., address, building number, and room number) for the subsystem architecture assessment meeting.

      Rationale: Attendees need an exact time to avoid schedule conflicts.

c. Develop meeting agenda.

Substep: The assessment team and subsystem architecture team set up a meeting agenda that covers introducing the subsystem, reviewing the requirements, introducing the architecture, presenting and probing the individual quality cases, and wrapping up the meeting by assigning action items.

Rationale: Agendas are important to enable attendees to determine if they must attend only certain parts of the meeting (e.g., reliability, safety, and security specialty engineers may only need to attend during the presentation of the associated safety cases).

d. Invite stakeholders.

Substep: The assessment team sends invitations including the meeting agenda to the identified meeting attendees and other stakeholders.

Rationale: Documented invitations (e.g., email with attached agenda) are convenient for invitees who can add the meeting to their schedules and read the agendas.

Preparation Task Postconditions

This task is successfully completed when the following postconditions are met:

- The subsystem architecture team is ready for the subsystem architecture assessment meeting when
  - The subsystem architecture team has received the architecture assessment checklist.
  - The subsystem architecture team has generated architecture assessment preparatory and presentation materials.
- The assessment team is ready for the subsystem architecture assessment meeting when
  - The assessment team had obtained and read the preparatory and presentation materials generated by the subsystem architecture team.
  - The assessment team has submitted any preliminary RFIs and RFAs to the subsystem assessment team.
- The subsystem architecture assessment meeting is organized.

5.3.2 Subsystem Architecture Assessment – Meeting

Meeting Task Objectives

The objectives of this task are as follows:

- Present quality cases. The architecture team presents their quality cases to the assessment team.
- Probe architecture. The assessment team actively questions the architecture team to identify
  - architecture defects and weaknesses (both by commission and omission)
  - architectural risks

Meeting Task Duration
The duration of this task is typically one day per subsystem (or two hours per quality factor). However, the duration may vary depending on the

- size and complexity of the subsystem architecture
- number of quality cases presented
- number of architectural defects and risks found
- skill, expertise, and productivity of the teams involved

Meeting Task Preconditions
This task can be started when the following preconditions are met:

- The subsystem architecture team is ready for the subsystem architecture assessment meeting when
  - The subsystem architecture team has received the architecture assessment checklist.
  - The subsystem architecture team has generated architecture assessment preparatory and presentation materials.
- The assessment team is ready for the subsystem architecture assessment meeting when
  - The assessment team has obtained and read the preparatory and presentation materials generated by the subsystem architecture team.
  - The assessment team has submitted any preliminary RFIs and RFAs to the subsystem assessment team.
- The subsystem architecture assessment meeting is organized.

Meeting Task Steps
During each architecture assessment meeting, perform the following steps:

1. **Introduce the subsystem.**

   Step: The subsystem architecture team presents to the assessment team a brief introduction to the subsystem. This introduction includes the
   - primary purpose of the subsystem
   - placement of the subsystem is in the overall hierarchy of the system architecture (e.g., composition diagram)
   - context of the subsystem, including the other subsystems with which it interfaces (e.g., a context diagram)
   - primary functions that are allocated to the subsystem

   Rationale: This step helps to set the stage for the following steps and ensures that the assessment team members truly understand the basics of the subsystem.

2. **Review the requirements.**

   Step: The subsystem architecture team presents to the assessment team a brief review of the architecturally significant quality requirements that drive the architecture of the subsystem. They concentrate on the

- quality factors important to the subsystem including their relative priorities and conflicts
- associated goals that have been derived and allocated to the subsystem
- critical quality requirements

Rationale: This step reminds everyone of the quality factors driving the architecture decisions made by the architecture team. It provides an overview of the content of the quality case claims and the motivation for the quality case arguments.

3. Introduce the architecture.

Step: The subsystem architecture team presents to the assessment team a brief introduction of the architecture of the subsystem including the

- main components of the subsystem including name and purpose
- major relationships between these components of the subsystem and with external subsystems, systems, and users
- overview of the most important architecture decisions and their rationales
- major engineering tradeoffs made to support conflicting quality factors

Rationale: If members of the assessment team do not get an overview of the architecture and its key decisions before delving into the individual quality cases, there is a danger of "missing the forest for the trees." Because individual quality cases are specific to individual quality factors, this is the appropriate step for assessing the tradeoffs the architects had to make between conflicting quality factors (e.g., increasing security may decrease performance). This is also when the assessment team determines if the architects' "story" of the architecture is cohesive and consistent.

The remaining steps are performed once for each quality factor on a quality case-by-quality case basis.

1. Present individual quality cases.

Step: The subsystem architecture team presents to the assessment team their individual quality cases, including the following components for each quality case:

a. Brief Summary

A brief summary of the quality case (e.g., quality case diagram)

b. Claims

The architects' claims consist of quality goals and requirements sufficiently supported by their architecture decisions. Goals can be that the architecture provides a sufficient amount of a specific quality factor (e.g., performance) or quality subfactor (e.g., response time or throughput). Requirements should be explicitly stated rather than merely referencing requirements identifiers.

c. Arguments

The architects present clear and compelling arguments that their architecture justifies belief in their claims. Arguments consist of their architectural decisions and associated rationales. Rationales should also include any specific engineering tradeoffs they made to support conflicting quality factors.

d. Evidence

The architects present official evidence supporting their arguments. Evidence typically consists of diagrams and references to relevant parts of models and documents. Evidence can also consist of demonstrations witnessed by members of the assessment team.[42]

Rationale: This is the key step of the architecture assessment when the architecture team attempts to convince the assessment team that their architecture is good enough to pass the assessment. It presents the quality cases in a logical structure that emphasizes its key components and their relationships.

2. Probe the architecture.

Step: The assessment team probes the architecture and quality cases, looking for architecture defects and weaknesses (both by commission and omission), architecture risks, and potential areas of improvement. Based on their reading the preparatory materials and the presentations made during the meeting, the assessment team asks members of the subsystem architecture team probing questions to

- clarify or expand on their arguments
- display specific evidence backing up their arguments
- clarify engineering tradeoffs made to support quality requirements for conflicting quality factors
- dive deeper into certain areas of the architecture (e.g., lower level tiers and subsystems)
- use one or more scenarios as test cases to test the architecture's support for a quality requirement
- provide additional arguments and evidence[43]

Rationale: A natural tendency for the subsystem architecture team is to concentrate on the strengths of their architecture when presenting quality cases. It is important for the assessment team to help the subsystem architecture team identify any defects and weaknesses in their architecture early when they are easier and less expensive to fix. This is especially true of missing architecture decisions (i.e., omission), which are easy for the subsystem architecture team to overlook.

3. Manage action items.

Step: The assessment team collaborates with the subsystem architecture team to collect, identify, and record action items. This includes setting due dates and assigning the action items to appropriate people. Typical meeting action items include

[42] Demonstrations often occur later, for instance during a visit to a development laboratory.

[43] Depending on the additional information requested, the architecture team may be able to provide this information immediately, later during the meeting, or during the following Follow-Through task.

- Members of the subsystem architecture team are assigned RFIs and RFAs by the assessment team (e.g., finding and providing missing arguments and evidence to the assessment team).
- The assessment team members are assigned the tasks of supplying their assessment notes to the assessment team scribe.
- The assessment team leader is assigned the task of producing the assessment outbrief.
- The assessment team members are assigned the task of reviewing the outbrief before it is presented to the subsystem architecture team and any other meeting attendees or stakeholders (e.g., managers, members of the top-level architecture team, and members of the subsystem requirements team).
- The assessment team leader presents the outbrief to the meeting attendees and other interested stakeholders.
- Assessment team members are assigned the tasks of producing, reviewing, and distributing the subsystem architecture assessment report.

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and tracked to completion.

Meeting Task Postconditions
This task is successfully completed when the following postconditions are met:

- The subsystem architects have presented an introduction of their subsystem to the assessment team.
- The subsystem architects have presented a review of the relevant quality requirements to the assessment team.
- The subsystem architects have presented an introduction of their subsystem architecture to the assessment team.
- The subsystem architects have presented their quality cases to the assessment team.
- The assessment team has probed the subsystem architecture.
- Action items have been captured, assigned, and scheduled.

5.3.3 Subsystem Architecture Assessment – Follow-Through

Follow-Through Task Objectives
The objectives of this task are to

- **Develop a Consensus**

  The assessment team develops a consensus regarding the quality of the subsystem architecture.

- **Present a Meeting Outbrief**

  The assessment team presents an outbrief of the assessment results to the meeting attendees.

- **Produce Final Subsystem Architecture Assessment Report**

  The assessment team produces a final version of the subsystem architecture assessment report.

- **Track Action Items**

  Action items from the subsystem architecture assessment are tracked to completion.

- **Capture Lessons Learned**

  Lessons learned regarding the subsystem architecture assessment phase are captured.

- **Update Assessment Method and Training Materials**

  The lessons learned are used to update the system architecture assessment method and its associated training materials.

Follow-Through Task Duration
The duration of this task usually a few weeks, with the majority of the effort spent on the development of the assessment report.

Follow-Through Task Preconditions
This task can be started when the following preconditions are met:

- The subsystem architects have presented an introduction of their subsystem to the assessment team.
- The subsystem architects have presented a review of the relevant quality requirements to the assessment team.
- The subsystem architects have presented an introduction of their subsystem architecture to the assessment team.
- The subsystem architects have presented their quality cases to the assessment team.
- The assessment team has probed the subsystem architecture.
- Action items have been captured, assigned, and scheduled.

Follow-Through Task Steps
After each subsystem architecture assessment meeting, the following steps are performed in a timely manner:

1. **Pool, discuss, and obtain consensus on observations and recommendations.**

   Step: Once the subsystem architecture assessment meeting is over and the architects have left the room, the meeting attendees on a quality case-by-quality case basis use their meeting notes as a basis to pool and discuss their observations and recommendations.

   Rationale: It is important for the meeting attendees to discuss their observations and recommendations while they are still fresh in everyone's minds. This enables them to develop a consensus concerning how well the architecture assessed supports its allocated

quality-related requirements. This, in turn, forms the basis for the contents of the meeting outbrief and report.

2. **Produce, review, and present subsystem architecture assessment outbrief.**

   Step: Within a day or two of the subsystem architecture assessment meeting and before leaving the site of the meeting:

   a. The members of the assessment team (especially the assessment team scribe) provide the leader of the assessment team with meeting notes.
   b. The leader of the assessment team produces an initial version of the subsystem architecture assessment outbrief.
   c. The assessment team members review the outbrief to ensure that it correctly summarizes the results of the subsystem architecture assessment.
   d. The leader of the assessment team iterates the subsystem architecture assessment outbrief to incorporate the comments and recommendations of the assessment team members.
   e. The leader of the assessment team presents the outbrief to the attendees and available stakeholders of the subsystem architecture assessment.

   Rationale: It is important to provide an informal outbrief at the end of the subsystem architecture assessment meeting so that attendees and stakeholders need not wait several weeks to obtain a clear indication of the results of the meeting.

3. **Produce, review, and distribute subsystem architecture assessment report.**

   Step: Within a couple of weeks of the subsystem architecture assessment meeting, complete the following steps:

   a. Members of the assessment team provide their notes to the members of the team tasked to produce the subsystem architecture assessment report.[44]
   b. Selected members of the assessment team produce an initial version of the subsystem architecture assessment report.
   c. The assessment team members review the subsystem architecture assessment report to ensure that it correctly records the results of the subsystem architecture assessment.
   d. The leader of the assessment team iterates the subsystem architecture assessment report to incorporate the comments and recommendations of the assessment team members.
   e. The assessment team distributes the subsystem architecture assessment report to its stakeholders.

   Rationale: This is the primary output of this phase and forms the basis of the system architecture quality assessment summary report.

[44] A subset of the assessment team may be tasked to produce the entire subsystem architecture assessment report. On the other hand, the assessment team may be divided into pairs which are tasked to produce different subsections of the meeting report. The choice of approach may vary depending on the size and formality of the subsystem architecture assessment meeting report, which could contain a significant number of observations and recommendations.

4. **Track action items to completion.**

   Step: The assessment team tracks open action items to completion.

   Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and tracked to completion.

5. **Capture lessons learned.**

   Step: The assessment team collaborates with the subsystem architecture team to capture lessons learned about the effectiveness of the subsystem architecture assessment phase of the QUASAR method.

   Rationale: This enables the method to be continually improved and tailored for future subsystem assessments and use on future projects.

6. **Update assessment method and associated training materials.**

   Step: Based on the lessons learned, the project process team[45] updates the project-tailored QUASAR procedure and the associated QUASAR training materials. Where appropriate, they also provide the SEI with the lessons learned and updated procedure and training materials so that the official QUASAR documentation can be iterated.

   Rationale: Updating the QUASAR method enables the process to be improved on future subsystem architecture assessment phases and future projects.

Follow-Through Task Postconditions
This task is successfully completed when the following postconditions are met:

- The assessment team has presented the subsystem architecture assessment outbrief to attendees of the meeting.
- The assessment team has completed the subsystem architecture assessment report and distributes it to its stakeholders.
- Action items from the meeting have been tracked to completion.
- Lessons learned about the assessment method have been incorporated into the method and associated training materials.

5.4 System Architecture Assessment Summary Phase

Depending on the approach chosen to summarize the results of the overall architecture quality assessment, a system architecture assessment summary phase occurs

- once, at the end of the QUASAR assessment

[45] If an official “project process team” (e.g., system engineering process group [SEPG]) does not exist, then whichever team, group, or individual that is filling the process engineer role should perform this task.

- incrementally, before major milestone reviews (in order to feed into these reviews)
- in an ongoing, incremental manner as the quality of the subsystem architectures are assessed

Phase Objectives
The objectives of this phase are to

- Collect the results of all of the preceding subsystem architecture quality assessments and document the resulting overall quality assessment of the system's architecture.
- Perform a final outbrief capturing what went well and what could be improved regarding the system architecture quality assessment process.

Phase Tasks
As illustrated in Figure 15 on page 85, the system architecture assessment summary phase consists of the following three tasks, which are performed sequentially:

1. **System Architecture Assessment Summary – Preparation**
2. **System Architecture Assessment Summary – Initial Kickoff Meeting**
3. **System Architecture Assessment Summary – Follow-Through**

Summarizing the Results of the Individual Subsystem Architecture Assessments
There are three basic ways that the results of the individual subsystem architecture quality assessments can be summarized into a final assessment of the quality of the overall system. Therefore, one of the goals of tailoring the QUASAR method is to select one or more of the most appropriate of these summarization approaches based on the needs of the organizations involved. Each approach to summarizing the results of the individual subsystem architecture quality assessments has its own advantages and disadvantages. The selected summarization approaches should meet the specific needs of the stakeholder organizations and best address the primary reasons for performing the architecture assessments. An approach should be selected from the following list:

1. **Average of Subsystem Architecture Quality**

   When using this approach, the quality of the system architecture is the [weighted] average of the qualities of its subsystem architectures. Assume that the assessment team has already used their expert judgment to assign a color value (e.g., green, yellow, and red) to the subsystem architecture's support for each individual quality factor's set of associated quality requirements (e.g., a subsystem's architectural support for interoperability is green, reliability is yellow, and performance is red). The assessment team then assigns each color a standard numerical value (e.g., green = 2, yellow = 1, and red = 0). If appropriate, the assessment team assigns each quality factor a numerical weight based on the criticality of that quality factor to the subsystem's architecture and also assigns a

numerical weight to the criticality of that subsystem's architecture to the quality of the overall architecture of the system.[46]

a. Advantages

- When properly defined for the entire set of subsystem architecture assessments, the averaging approach provides a standard way to compare the results of multiple subsystem architecture quality assessments.
- This approach provides a simplified answer to the question: "What is the quality of the system's architecture?"

b. Disadvantages

- This approach attempts to build an "objective" numerical structure on top of a foundation of "subjective" expert opinion (a Delphi approach). It "mixes apples and oranges" in a statistically highly questionable manner.
- Although this approach provides the appearance of numerical legitimacy, the resulting value it provides is probably far less accurate and precise than it seems.
- It is extremely difficult to quickly and easily set the numerical color values and weightings in a manner that supports the goals of the assessment. This can become a major source of disagreement and wasted time.
- This approach is by far the most complex and labor intensive to use.
- Useful information is lost because this approach does not clearly identify those parts of the overall system architecture that still need work, what kind of problem exists (e.g., safety, security, usability), or even the severity of the problem.
- It is highly questionable if the extra effort needed to provide this kind of average assessment is worth the questionable value derived given its potential disadvantages.

c. Appropriateness

This approach is *not* recommended. If used, its use should probably be restricted to organizational internal assessments, the goal of which is to obtain a rough indication of the overall quality of the system architecture.

2. Worst Subsystem Architecture Quality

In this approach, the quality of the overall system architecture is equated with the worst quality subsystem architecture. Assign the worst value of any subsystem architecture quality assessment as the value of all of its super-systems including the overall system. Thus, if the quality of any subsystem's architecture for any cohesive set of quality requirements (i.e., associated with any quality factor) is red, then the quality of the overall system architecture is also red.

a. Advantages

- The approach is objective and easy to apply.
- This approach provides a great incentive for the architects to fix any problems that prevent a subsystem's architecture from adequately supporting a quality requirement.

[46] Note that the definitions for colors and weightings should be well-defined and standardized across all subsystems of the system.

- It provides a simple answer to the question: "What is the quality of the system's architecture?"

b. Disadvantages

- This approach denies the architects credit for all of the good architectural decisions they have made; a single subsystem's architecture failing to adequately support a single quality requirement is sufficient cause for the entire system architecture to "fail" the assessment.
- This approach does not take into account that all quality requirements and all subsystem architectures are not equally critical.

c. Appropriateness

This approach is probably best used for assessing contract compliance because one *could* argue that if a subsystem's architecture does not adequately support a quality requirement, then the overall system's architecture also does not adequately support that same quality requirement.

3. Union of Subsystem Architecture Qualities

With this summarization approach, the quality of the overall system architecture is the union of the qualities of its subsystem architectures. Collect and collate the information from all of the subsystem architecture quality assessments into one or more matrices that shows a mapping from the subsystem (i.e., matrix row) and quality factor (i.e., matrix column) to its associated quality representing color (i.e., associated cell in the matrix).

a. Advantages

- The approach is objective and relatively easy to apply.
- This approach clearly identifies those parts of the overall system architecture that still need work, what kind of problem exists (e.g., safety, security, usability), or the severity of a problem.
- This approach provides full credit for all of the good architectural decisions that the architects made; a single subsystem's architecture failing to adequately support a single quality requirement is *not* sufficient to cause the entire architecture to "fail" the assessment.
- This approach takes into account the fact that all quality requirements and all subsystem architectures are not equally critical.
- This approach provides an ongoing, iterative, and incremental summary of the system architecture as the qualities of its subsystem architectures are assessed.
- No information is lost in producing a single overall numerical (or color) value for the quality of the overall system architecture.
- This approach can be easily combined with the previous approaches.

b. Disadvantages

- This approach does not provide a single, simple answer to the question: "What is the quality of the system's architecture?"[47]
- This approach may not be acceptable to managers who demand such a single answer to the question.

[47] Then again, there may not be any reasonable single answer to that question.

– On really large systems in which the architectures of many subsystems are assessed, the result of this approach becomes somewhat more difficult to evaluate as the scope of the assessment increases in numbers of subsystems and numbers of quality factors.

c. Appropriateness

This approach should probably always be used, regardless of whether any of the preceding approaches are also used. This approach should definitely be used if the primary reason for performing the architecture quality assessments is to determine how to allocate limited resources for improving the subsystem architectures.

5.4.1 System Architecture Assessment Summary – Preparation

Preparation Task Objective

The objectives of this task are to

- Gather the results of the quality assessments of the individual subsystem architectures that are within the scope of the overall system architecture's quality assessment.
- Organize and summarize the results in preparation for the system architecture assessment summary meeting.

Preparation Task Duration

The duration of this task varies depending on whether the final phase occurs

- once, at the end of all of the individual subsystem architecture quality assessments
- prior to each major project milestone meeting during which the state of the system architecture is presented
- on an ongoing basis as the individual subsystem architecture quality assessments are performed

Preparation Task Preconditions

This task can be started when all (or a sufficient number for an intermediate system architecture assessment summary) of the subsystem architecture assessments have been completed.

Preparation Task Steps

The following steps are performed during this task:

1. Collect individual subsystem architecture quality assessments.

Step: The assessment team collects the results of the individual subsystem architecture quality assessments.

Rationale: The overall system architecture quality assessment is a summation of the individual subsystem architecture quality assessments.

2. **Summarize subsystem architecture quality assessment results.**

   Step: Using the agreed-upon summarization process, the assessment team organizes and summarizes the individual subsystem architecture quality assessments results in preparation for the system architecture assessment summary meeting. This includes creating the subsystem summary matrix.

   Rationale: It is important to use the consensus approach for summarizing the individual subsystem architecture quality assessment results.

3. **Identify primary stakeholders.**

   Step: The assessment team collaborates with the system architecture team to identify the primary stakeholders who may

   a. attend the system architecture assessment summary meeting
   b. receive a copy of the system architecture assessment summary report

   Rationale: It is important to invite all relevant stakeholders to the meeting and provide them with a copy of the presentation materials and summary report so that no significant stakeholder is excluded.

4. **Produce, review, and distribute the architecture assessment summary report.**

   Step: In the weeks leading up to the system architecture quality assessment meeting, the following substeps are performed:

   a. Produce the architecture assessment summary report.

   Substep: Based on the individual subsystem architecture quality assessment results and their summaries, selected members of the assessment team produce the system architecture quality assessment summary report for the system architecture quality assessment meeting.

   Rationale: The summary report enables is the primary deliverable of the system architecture assessment.

   b. Perform an internal review of the architecture assessment summary report.

   Substep: The members of the assessment team perform an internal quality check on the system architecture quality assessment summary report.

   Rationale: It is cost-effective to identify and fix defects prior to distribution of the report.

   c. Perform an internal review of the architecture assessment summary report.

   Substep: The authors of the system architecture quality assessment summary report make any final fixes prior to initial distribution.

   Rationale: Fixing mistakes prior to the meeting enables the attendees to concentrate on the content of the system architecture quality assessment summary report rather than any defects it might contain.

   d. Distribute the architecture assessment summary report.

   Substep: The authors of the system architecture quality assessment summary report distribute the initial version to the meeting attendees and other stakeholders.

Rationale: This enables the meeting attendees to read the system architecture quality assessment summary report prior to the meeting.

5. Produce, review, and distribute the summary presentation materials.

Step: In the weeks leading up to the system architecture quality assessment meeting, the following substeps are performed:

a. Produce summary presentation materials.

Substep: Based on the system architecture quality assessment summary report, selected members of the assessment team produce the system summary meeting presentation materials for the system architecture quality assessment meeting.

Rationale: The presentation materials enable an overview of the content of the system architecture quality assessment summary report to be distributed and presented at the meeting.

b. Perform an internal review of the summary presentation materials.

Substep: The members of the assessment team perform an internal quality check on the system summary meeting presentation materials.

Rationale: It is cost-effective to identify and fix defects prior to distribution and presentation of the meeting materials.

c. Perform an internal review of the summary presentation materials.

Substep: The authors of the system summary meeting presentation materials make any final fixes prior to distribution.

Rationale: Fixing mistakes prior to the meeting enables the attendees to concentrate on the content of the presentation materials rather than defects.

d. Distribute the summary presentation materials.

Substep: The authors of the system summary meeting presentation materials distribute the final presentation materials to the meeting attendees and other stakeholders.

Rationale: This enables the meeting attendees to read the presentation materials prior to the meeting.

6. Organize the meeting.

Step: At least two weeks prior to the system architecture assessment summary meeting, the assessment team and system architecture team collaborate to complete the following steps:

a. Identify meeting attendees and other stakeholders.

Substep: The assessment and system architecture teams develop a list of meeting attendees and other stakeholders who should receive copies of the system architecture summary report and meeting presentation materials.

Rationale: It is important to ensure that no significant stakeholder is overlooked.

b. Set time and location.

Substep: The assessment and system architecture teams determine an exact time and location (e.g., address, building number, and room number) for the system architecture assessment summary meeting.

Rationale: Attendees need exact times to avoid conflicts.

c. Develop meeting agenda.

Substep: The assessment team and system architecture team set up a meeting agenda covering introducing restatement of assessment objectives, summary of assessment method, summary of quality of the architecture of the individual subsystems, summary of the overall system architecture, and meeting wrap-up (e.g., status of action items and assignment of final new action items).

Rationale: Agendas enable attendees to identify the most important parts of the meeting.

d. Invite stakeholders.

Substep: The assessment team sends an invitation including the meeting agenda to the identified meeting attendees and other stakeholders.

Rationale: Documented invitations (e.g., email with attached agenda) are convenient for invitees who can add the meeting to their schedules and read the agendas.

Preparation Task Postconditions

This task is successfully completed when the following postconditions are met:

- Individual subsystem architecture assessment reports have been gathered and their results have been summarized.
- The initial version of the system architecture quality assessment summary report has been produced, reviewed, and distributed to its primary stakeholders.
- The presentation materials have been produced, reviewed, and distributed to the stakeholders.
- The system architecture assessment summary meeting has been organized.

5.4.2 System Architecture Assessment Summary – Meeting

Meeting Task Objectives

The objectives of this task are to

- Present the overall results of the system architecture quality assessment to its major stakeholders.
- Perform a final review[48] on the system architecture quality assessment process.

Meeting Task Duration

The duration of this task is typically half a day.

[48] Such a review is often referred to as a “postmortem.”

Meeting Task Preconditions
This task can be started when the following preconditions are met

- Individual subsystem architecture assessment reports have been gathered and their results have been summarized.
- The initial version of the system architecture quality assessment summary report has been produced, reviewed, and distributed to its primary stakeholders.
- The presentation materials have been produced, reviewed, and distributed to the stakeholders.
- The system architecture assessment summary meeting has been organized.

Meeting Task Steps
During this task, the following steps are performed:

1. **Restate assessment objectives.**

   Step: The assessment team presents a restatement of the overall system architecture quality assessment objectives.

   Rationale: Not all attendees at the final meeting, especially members of upper management, will be familiar with the specific objectives of the method used to arrive at the overall assessment of the quality of the system's architecture.

2. **Summarize assessment method.**

   Step: The assessment team presents to the meeting attendees a brief summary of the [tailored] method used to assess the quality of the system architecture.

   Rationale: Not all attendees at the final meeting, especially members of upper management, will be familiar with the method used to arrive at the overall assessment of the quality of the system's architecture.

3. **Summarize quality of subsystem architectures.**

   Step: The assessment team presents an overview of the *subsystem* architecture quality assessment results (to date if this final phase is performed incrementally).

   Rationale: The subsystem architectures are the foundation on which the overall system architecture is built. It is useful to have a brief summary of the preceding assessments as they are likely to have occurred over the course of several months (or even years) and are not fresh in the minds of most of the meeting attendees.

4. **Present summary of the quality of the system architecture.**

   Step: The assessment team presents a summary of the *system* architecture quality. This may be the quality of subsystem architectures assessed to date if this final phase is performed incrementally.

   Rationale: This step presents the overall results of the QUASAR method, which is the primary result of the overall assessment method.

5. **Solicit feedback.**

   Step: The assessment team solicits comments concerning the assessment results, especially presentation or report contents that are factually incorrect.

   Rationale: This step enables all stakeholders to have a final chance to ensure the quality of the system architecture assessment report.

6. **Capture lessons learned.**

   Step: The assessment team collaborates with meeting attendees to capture lessons learned about the effectiveness of the system architecture assessment summary phase of the QUASAR method.

   Rationale: This step enables the method to be continually improved and tailored for future projects.

Meeting Task Postconditions
This task is successfully completed when the following postconditions are met:

- The objectives of the system architecture quality assessment have been presented to the meeting attendees.
- A summary of the system architecture quality assessment method has been presented to the meeting attendees.
- The quality of each of the subsystem architectures has been presented to the meeting attendees.
- The summary of the overall quality of the system architecture has been presented to the meeting attendees.
- Feedback has been solicited from the meeting attendees.
- Lessons learned about the system architecture quality assessment method have been captured.
- The members of the assessment team have taken notes.

5.4.3 System Architecture Assessment Summary – Follow-Through

Follow-Through Task Objective
The objective of this task is to finalize the work associated with the system architecture quality assessment.

Follow-Through Task Duration
Updating and distributing the final report should not take more than one or two weeks, depending on the number of changes to be made resulting from feedback at the final summary meeting. Similarly, the amount of time taken to track action items to completion varies depending on the number and types of action items. Finally, updating the assessment method's

training materials and associated procedure largely depends on the number and types of lessons learned.

Follow-Through Task Preconditions
This task can be started when the following preconditions are met:

- The objectives of the system architecture quality assessment have been presented to the meeting attendees.
- A summary of the system architecture quality assessment method has been presented to the meeting attendees.
- The quality of each of the subsystem architectures has been presented to the meeting attendees.
- The summary of the overall quality of the system has been presented to the meeting attendees.
- Feedback has been solicited from the meeting attendees.
- Lessons learned about the system architecture quality assessment method have been captured.
- The members of the assessment team have taken notes.

Follow-Through Task Steps
After the system architecture assessment summary meeting, the following steps are performed in a timely manner:

1. **Update the system architecture assessment summary report.**

   Step: The assessment team makes any final updates to the system architecture assessment summary report based on inputs from the system architecture assessment summary meeting.

   Rationale: This step produces the final version of primary deliverable of the QUASAR method.

2. **Distribute the system architecture assessment summary report.**

   Step: The assessment team distributes the final version of the system architecture assessment summary report to its stakeholders.

   Rationale: This step ensures that the stakeholders receive the results of the assessments and thereby realize the value of the preceding steps (i.e., obtain the return on their investment in prior effort).

3. **Manage action items.**

   Step: The assessment team collaborates with the meeting attendees to collect, identify, and record any action items from the meeting. This includes setting due dates and as-

signing the action items to appropriate people. The assessment team tracks all remaining action items to closure.

Rationale: Unless action items are assigned and due dates are scheduled, the action items are unlikely to be properly handled in a timely manner and tracked to completion.

4. **Update assessment method and associated training materials.**

   Step: Based on the lessons learned, the project process team[49] updates the project-tailored QUASAR procedure and the associated QUASAR training materials. Where appropriate, they also provide the SEI with the lessons learned and updated procedure and training materials so that the official QUASAR documentation can be iterated.

   Rationale: Updating the QUASAR enables the process to be improved on future system architecture assessment summary phases and future projects.

Follow-Through Task Postconditions

This task is successfully completed when the following postconditions are met:

- The final system architecture assessment summary report has been produced and distributed to its stakeholders.
- Action items have been tracked to completion.
- The system architecture quality assessment procedure and associated training materials have been updated.

[49] If an official "project process team" (e.g., system engineering process group [SEPG]) does not exist, then whichever team, group, or individual that is filling the process engineer role should perform this task.

6 QUASAR Work Products

The purpose of the QUASAR method is to assess the quality of system architecture, not to produce documentation just for the sake of having it. Nevertheless, some documentation can be very useful for recording and conveying important information related to the assessment of system architectures. Where appropriate and cost-effective, the teams performing the tasks of the QUASAR method typically produce the following work products:

1. **System Architecture Assessment Initiation Work Products**
    a. Architecture Assessment Procedure
    b. Architecture Assessment Training Materials
    c. Initial Kickoff Meeting Agenda
    d. Initial Kickoff Meeting Assessor Notes
    e. Initial Kickoff Meeting Minutes
    f. Assessment Schedule
    g. Assessment Action Item List
2. **Subsystem Requirements Review Work Products**
    a. Subsystem Requirements Review Checklist
    b. Subsystem Requirements Review Preparatory Materials
    c. Subsystem Requirements Review Presentation Materials
    d. Subsystem Requirements Trace
    e. Subsystem Requirements Review Meeting Agenda
    f. Subsystem Requirements Review Meeting Assessor Notes
    g. Subsystem Requirements Review Meeting Outbrief
    h. Subsystem Requirements Review Meeting Minutes
    i. Updated Assessment Action Item List
3. **Subsystem Architecture Assessment Work Products**
    a. Subsystem Architecture Assessment Checklist
    b. Subsystem Architecture Assessment Preparatory Materials
    c. Subsystem Architecture Assessment Presentation Materials
    d. Subsystem Architecture Assessment Meeting Agenda
    e. Subsystem Architecture Assessment Meeting Assessor Notes
    f. Subsystem Architecture Support Matrix
    g. Subsystem Architecture Assessment Meeting Outbrief
    h. Subsystem Architecture Assessment Report
    i. Updated Assessment Action Item List

4. **System Architecture Assessment Summary Work Products**
    a. System Summary Subsystem Matrix
    b. System Summary Meeting Presentation Materials
    c. System Architecture Assessment Summary Meeting Agenda
    d. System Architecture Assessment Summary Meeting Assessor Notes
    e. System Architecture Quality Assessment Summary Report

6.1 System Architecture Assessment Initiation Work Products

As illustrated in Figure 16, the following work products are produced for and during the system architecture assessment initiation phase and are described in detail in the sections that follow:

1. Architecture Assessment Procedure
2. Architecture Assessment Training Materials
3. Initial Kickoff Meeting Agenda
4. Initial Kickoff Meeting Assessor Notes
5. Initial Kickoff Meeting Minutes
6. Assessment Schedule
7. Assessment Action Item List

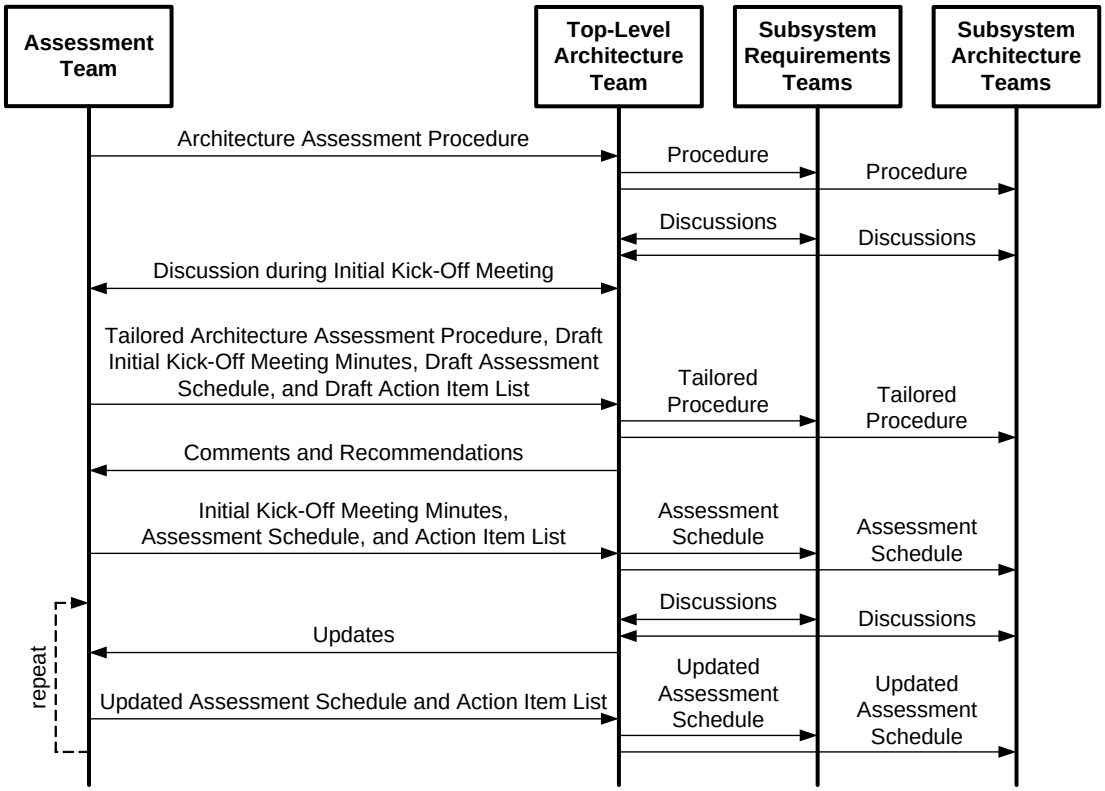

Figure 16: Initial Kickoff Meeting Work Products

6.1.1 Architecture Assessment Procedure

- **Definition -** the organizational or system-specific, tailored procedure that documents how to perform the architecture assessments
- **Objectives**

  Document the system architecture quality assessment method, including the
  - tasks to be performed during the assessments
  - teams that will perform these tasks
  - work products to be produced by these teams
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the
    - assessment team (prior to delivery to the architecture team)
    - top-level architecture team (prior to the initial kickoff meeting)
    - requirements teams (prior to the subsystem requirements review meetings)
    - subsystem architecture teams (prior to the subsystem architecture assessment meetings)
  - Maintained by the assessment team

  - Used by the
    - architecture teams to understand their responsibilities including tasks to perform and work products to produce
    - assessment team to understand their responsibilities including tasks to perform and work products to produce
    - management team to understand the resources required to implement the assessment method
    - requirements teams to understand their responsibilities including tasks to perform and work products to produce

- **Inputs**
  - default assessment procedure (i.e., this handbook)
  - attendees' prior experience performing architecture assessments
- **Contents**
  - Front Matter
  - Introduction
    - Objectives of the Assessments
    - Terms and Concepts
  - Quality Cases
    - Potentially Relevant Quality Factors
    - Quality Goals and Requirements
    - Claims
    - Arguments
    - Evidence
    - Examples
  - Tasks
    - Objectives
    - Steps
    - Preconditions and Postconditions
  - Teams and Member Roles
    - Responsibilities
    - Team composition
  - Work Products
    - Name and Definition
    - Objectives (purpose)
    - Stakeholders
    - Inputs
    - Contents

6.1.2 Architecture Assessment Training Materials

- **Definition -** the organizational or system-specific, tailored training materials for teaching the architecture quality assessment method

- **Objectives**

  Enable the assessment team to train the assessment participants (e.g., new members and members of the development organization [members of the system and subsystem requirements and architecture teams]) how to perform the assessment method.

- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the
    - assessment team (prior to delivery to the architecture team)
    - top-level architecture team (prior to the initial kickoff meeting)
    - requirements teams (prior to the subsystem requirements review meetings)
    - subsystem architecture teams (prior to the subsystem architecture assessment meetings)
  - Maintained by the assessment team
  - Used by the
    - architecture teams to understand their responsibilities including tasks to perform and work products to produce
    - assessment team to understand their responsibilities including tasks to perform and work products to produce
    - management team to understand the resources required to implement the assessment method
    - requirements teams to understand their responsibilities including tasks to perform and work products to produce

- **Input**
  - Architecture Assessment Procedure

- **Contents**
  - Introduction
    - Objectives of the Assessments
    - Terms and Concepts
  - Quality Cases
    - Potentially Relevant Quality Factors
    - Quality Goals and Requirements
    - Claims
    - Arguments
    - Evidence
    - Examples
  - Tasks
    - Objectives
    - Steps
    - Preconditions and Postconditions
  - Teams and Member Roles
    - Responsibilities
    - Team Composition

- Work Products
  - Name and Definition
  - Objectives (purpose)
  - Stakeholders
  - Inputs
  - Contents

6.1.3 Initial Kickoff Meeting Agenda

- **Definition -** the informal, typically one-page agenda for the initial system-wide kickoff meeting
- **Objective**

  Inform meeting attendees of meeting topics and times.
- **Stakeholders**
  - Produced by the
    - assessment team
    - system requirements team
    - system architecture team
  - **Not** reviewed
  - **Not** maintained
  - Used by the meeting attendees
- **Inputs**
  - Architecture Assessment Procedure
  - discussions
- **Contents**
  - Meeting Topics and Times
    - Introductions and Meeting Logistics
    - Assessment Scope
    - Assessment Schedule
    - Tailoring of Assessment Method
    - Meeting Wrap-Up

6.1.4 Initial Kickoff Meeting Assessor Notes

- **Definition -** informal notes taken by an individual assessor during a subsystem requirements meeting
- **Objectives**
  - Capture information that the assessor considers significant.
  - Provide content for the subsystem requirements meeting outbrief and meeting minutes.

- **Stakeholders**
  - Produced by individual members of the assessment team who attend the subsystem requirements meeting
  - **Not** reviewed
  - **Not** maintained
  - Used by the scribe of the assessment team, to use as input to the subsystem requirements meeting outbrief and meeting minutes
- **Inputs**
  - Subsystem Requirements Meeting Checklist
  - Subsystem Requirements Meeting Preparatory Materials
  - Subsystem Requirements Meeting Presentation Materials
  - Subsystem Requirements Trace
  - Answers to assessors questions
- **Contents**
  - Informal Notes Capturing Observations or Key Findings
    - requirements driving the subsystem architecture
    - architects' representative partial quality cases
  - Recommendations for Improvement
    - requirements driving the subsystem architecture
    - architects' representative partial quality cases
    - architecture assessment method
  - Questions Asked and Answers Given
  - Other information that the individual assessor considered significant and worthy of note

6.1.5 Initial Kickoff Meeting Minutes

- **Definition -** the official record of the proceedings of the initial kickoff meeting
- **Objective**

  Record the major occurrences and decisions made during the initial kickoff meeting.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the meeting attendees (prior to publication)
  - Maintained by the assessment team (factual corrections only)
  - Used by the
    - architecture teams to understand consensus reached and agreements made
    - assessment team to understand consensus reached and agreements made
    - management team to understand required resources
    - requirements teams to understand consensus reached and agreements made
- **Input**
  - notes and comments from meeting attendees

- **Contents**
  - Meeting Date and Location
  - Invitees
    - Members of the Assessment Team
    - Members of the Top-Level Architecture Team
  - Scope of the Assessments
    - Subsystems to be Assessed
    - Prioritized List of Architecturally Significant Requirements (e.g., quality factors)
    - Assessment Resources (e.g., personnel and time)
  - Assessment-Specific Tailoring of the Assessment Method
  - Assessment Schedule (by reference – see Section 6.1.6)
  - Action Items (by reference – see Section 6.1.7)

6.1.6 Assessment Schedule

- **Definition -** the schedule that documents the tentative dates of the assessment meetings for the individual assessments and subsequent roll-up meetings
- **Objectives**
  - Enable scheduling of preparatory work (e.g., development of architecturally significant requirements, architecture documentation, and quality cases including claims, arguments, and evidence).
  - Enable scheduling of resources (e.g., personnel and meeting rooms).
  - Help avoid schedule conflicts (e.g., with major programmatic reviews and other development work).
- **Stakeholders**
  - Produced by the
    - assessment team
    - top-level architecture team
  - Reviewed by the
    - Architecture team (prior to publication)
    - Assessment team (prior to publication)
    - Management team (prior to publication)
  - Maintained by the assessment team (factual corrections only)
  - Used by the
    - architecture teams to understand schedule commitments
    - assessment team to understand schedule commitments
    - requirements teams to understand schedule commitments
    - management teams to understand schedule commitments
- **Input**
  - Initial Kickoff Meeting Minutes

- **Contents**
  - Agreed-Upon Dates of Initial Meetings
  - Tentative Dates for the Remaining Meetings

6.1.7 Assessment Action Item List

- **Definition -** a list of actions to be taken resulting from the meeting
- **Objective**

  Ensure that actions identified during the meeting are assigned and tracked to completion.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the meeting attendees
  - Maintained by the assessment team
  - Used by the attendees to keep track of assigned actions
- **Input**
  - Initial Kickoff Meeting Minutes
- **Contents**
  - Action Items
    - Unique Identifier
    - Action to be Taken
    - Person Assigned Action
    - Due Date
    - Status

6.2 Subsystem Requirements Meeting Work Products

As illustrated in Figure 17, the following work products are produced for and during the individual subsystem requirements meetings:

1. Subsystem Requirements Review Checklist
2. Subsystem Requirements Review Preparatory Materials
3. Subsystem Requirements Review Presentation Materials
4. Subsystem Requirements Trace
5. Subsystem Requirements Review Meeting Agenda
6. Subsystem Requirements Review Meeting Assessor Notes
7. Subsystem Requirements Review Meeting Outbrief
8. Subsystem Requirements Review Meeting Minutes
9. Action Item List (updated and maintained – see Section 6.1.7)

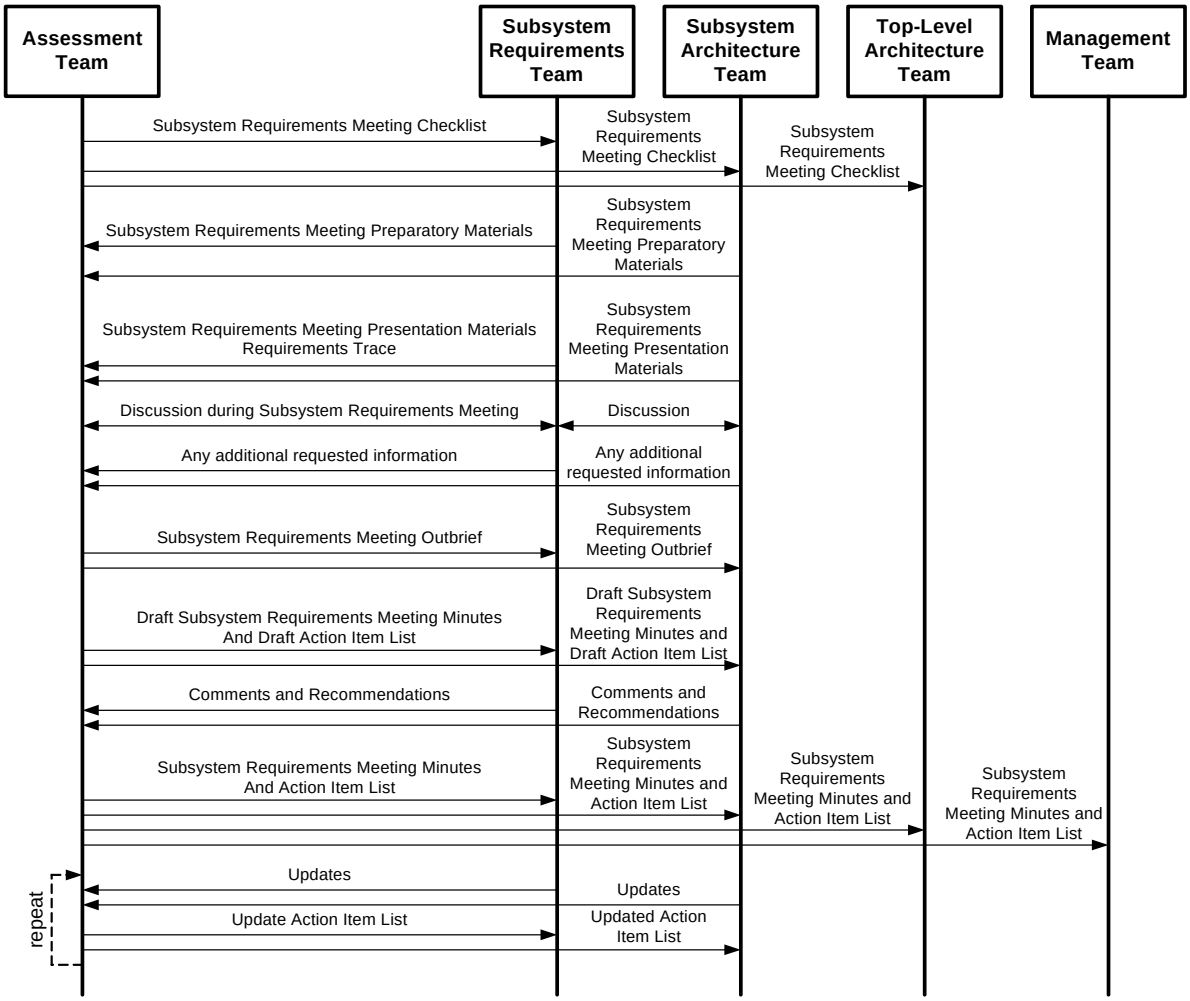

Figure 17: Subsystem Requirements Meeting Work Products

6.2.1 Subsystem Requirements Review Checklist

- **Definition -** a checklist that is used by the assessment team during the subsystem requirements meetings to help them identify defects in the requirements
- **Objectives**
  - Help the subsystem requirements team improve the engineering of the derived architecturally significant requirements allocated to the subsystem.
  - Help the subsystem architecture team better prepare for the subsystem architecture assessment meeting.
  - Help the assessment team identify defects associated with the architecturally significant requirements that have been allocated to the subsystem.
  - Help the assessment team identify defects associated with the subsystem architecture team's quality cases.

- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the
    - subsystem architecture team
    - subsystem requirements team
  - Maintained by the assessment team
  - Used by the
    - assessment team to identify defects associated with the architecturally significant requirements and the subsystem architecture team's representative samples of draft quality cases
    - subsystem architecture team to understand what is expected of them during the relevant subsystem requirements meeting and subsystem architecture assessment meeting
    - subsystem requirements team to understand what is expected of them during the subsystem requirements meeting
- **Inputs**
  - Architecture Assessment Method (tailored)
  - Default Subsystem Requirements Meeting Checklist (e.g., example in Appendix D in this handbook)
  - Initial Kickoff Meeting Minutes
- **Contents**
  - Requirements Questions
    - questions related to potential completeness defects associated with requirements driving the subsystem architecture
    - questions related to potential quality (ambiguity, feasibility, etc.) defects associated with requirements driving the subsystem architecture
  - Questions About Quality Cases
    - questions related to potential defects associated with claims
    - questions related to potential defects associated with arguments
    - questions related to potential defects associated with evidence

6.2.2 Subsystem Requirements Review Preparatory Materials

- **Definition -** materials provided to the assessment team prior to a subsystem requirements meeting
- **Objectives**
  - Enable the assessment team to properly prepare for the associated subsystem requirements meeting.
  - Enable the assessment team members to be more effective during the subsystem requirements meeting.
  - Decrease the time needed for the actual subsystem requirements meeting.

- **Stakeholders**
  - Produced by the
    - subsystem requirements team
    - subsystem architecture team
  - Reviewed by the
    - various teams, depending on the materials (requirements specifications, requirements repository, requirements traces, etc.)
    - top-level architecture team
  - Maintained by the
    - subsystem requirements team
    - subsystem architecture team
  - Used by the assessment team to become familiar with the content, quality, and status of the requirements that will significantly influence the subsystem architecture
- **Input**
  - varies depending on the materials
- **Contents**
  - Requirements-Related Information
    - requirements specifications with architecturally significant requirements identified (e.g., highlighted or indexed)
    - information providing read-only access to the requirements repositories
    - requirements traces with architecturally significant requirements highlighted or restricted to architecturally significant requirements (see below)
    - any questions that the subsystem requirements team might have regarding the pending subsystem requirements meeting
  - Architecture-Related Information
    - representative samples of incomplete draft quality cases
    - any questions that the subsystem architecture team might have regarding the future subsystem architecture assessment meeting

6.2.3 Subsystem Requirements Review Presentation Materials

- **Definition -** presentation materials that are presented to the assessment team during the subsystem requirements meeting
- **Objectives**
  - Communicate the architecturally significant requirements to the assessment team.
  - Communicate the subsystem architects' understanding of their responsibilities associated with the associated subsystem architecture assessment meeting.
- **Stakeholders**
  - Produced by the
    - subsystem requirements team
    - subsystem architecture team

  - Reviewed by the top-level architecture team
  - Maintained by the
    - subsystem requirements team
    - subsystem architecture team
  - Used by the assessment team to understand the requirements relevant to the subsystem architecture and the subsystem architect's understanding of their responsibilities associated with the associated subsystem architecture assessment meeting

- **Inputs**
  - Architecture Assessment Procedure
  - Initial Kickoff Meeting Minutes
  - Subsystem Requirements
  - Subsystem Requirements Meeting Preparatory Materials
  - feedback from the assessment team

- **Contents**
  - Requirements-Related Information
    - summary of the subsystem requirements relevant to the subsystem architecture
    - summary of the requirements traces with only the architecturally significant requirements or with the architecturally significant requirements highlighted (see Section 6.2.4)
    - remaining questions that the subsystem requirements team might have regarding the pending subsystem requirements meeting
  - Architecture-Related Information
    - representative, partial samples of architecture quality cases
    - remaining questions that the subsystem architecture team might have regarding the pending subsystem architecture assessment meeting

6.2.4 Subsystem Requirements Trace

- **Definition -** documentation tracing derived and allocated requirements that are relevant to the subsystem architecture back to their sources (e.g., contract requirements or requirements at the next higher tier in the architecture)

- **Objectives**
  - Ensure that all requirements that drive the subsystem architecture have been identified, derived, and allocated to the subsystem architecture.
  - Ensure that the requirements are of sufficient quality to enable the subsystem architects to properly develop the subsystem architecture.

- **Stakeholders**
  - Produced by the subsystem requirements team
  - Review varies depending on the requirements engineering method
  - Maintenance varies depending on the requirements engineering method

  - Used by the
    - assessment team to understand the requirements driving the subsystem architecture so that they can assess the subsystem architecture against these requirements
    - subsystem architecture team to drive the development of the subsystem architecture

- **Input**
  - varies depending on the requirements engineering method
- **Contents**
  - Mapping of requirements relevant to the subsystem architecture back to their sources (e.g., contract requirements or requirements at the next higher tier in the architecture)[50]

6.2.5 Subsystem Requirements Review Meeting Agenda

- **Definition -** the informal, typically one-page agenda for a single subsystem requirements review meeting
- **Objective**

  Inform meeting attendees of meeting topics and associated times.
- **Stakeholders**
  - Produced by the
    - assessment team
    - subsystem requirements team
    - subsystem architecture team
  - **Not** reviewed
  - **Not** maintained
  - Used by the meeting attendees
- **Inputs**
  - Architecture Assessment Procedure
  - discussions

[50] Depending on the availability of requirements management tools, this requirements trace may include all requirements allocated to the subsystem or only those requirements that drive the architecture of the subsystem.

- **Contents**
  - Meeting Topics and Times
    - Introductions and Meeting Logistics
    - Quality Factors
    - Quality Requirements
    - Sample Quality Cases
    - Special Assessment Staffing Needs
    - Schedule of Coming Events
    - Meeting Wrap-Up

6.2.6 Subsystem Requirements Review Meeting Assessor Notes

- **Definition -** informal notes taken by an individual assessor during a subsystem requirements meeting
- **Objectives**
  - Capture information that the assessor considers significant.
  - Provide content for the subsystem requirements meeting outbrief and meeting minutes.
- **Stakeholders**
  - Produced by individual members of the assessment team who attend the subsystem requirements meeting
  - **Not** reviewed
  - **Not** maintained
  - Used by the scribe of the assessment team, to use as input to the subsystem requirements meeting outbrief and meeting minutes
- **Inputs**
  - Subsystem Requirements Meeting Checklist
  - Subsystem Requirements Meeting Preparatory Materials
  - Subsystem Requirements Meeting Presentation Materials
  - Subsystem Requirements Trace
  - Answers to assessors' questions
- **Contents**
  - Informal Notes Capturing Observations or Key Findings
    - requirements driving the subsystem architecture
    - architects' representative partial quality cases
  - Recommendations for Improvement
    - requirements driving the subsystem architecture
    - architects' representative partial quality cases
    - architecture assessment method

- Questions Asked and Answers Given
- other information that the individual assessor considered significant and worthy of note

6.2.7 Subsystem Requirements Review Meeting Outbrief

- **Definition** - an interim top-level summary of the results of a subsystem requirements meeting
- **Objectives**
  - Communicate a summary of the assessment team's interim results from the subsystem requirements meeting.
  - Elicit comments and recommendations from the subsystem architecture and subsystem requirements teams, especially to correct any factual misunderstandings before they are incorporated into the meeting's minutes.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the
    - assessment team (internally before it is sent to the subsystem architecture and requirements teams)
    - subsystem architecture team
    - subsystem requirements team
  - Maintained by
    - Temporary document that is not maintained, but rather superseded by the subsystem requirements meeting minutes
  - Used by the
    - assessment team to communicate their interim results to the subsystem architecture team and requirements team
    - subsystem architecture team to understand the assessment team's interim observations, findings, and recommendations related to the architecture team's presentation
    - subsystem architecture team to understand the assessment team's interim observations, findings, and recommendations related to the requirements team's presentation
- **Inputs**
  - Subsystem Requirements Meeting Checklist
  - Subsystem Requirements Meeting Preparatory Materials
  - Subsystem Requirements Meeting Presentation Materials
  - Subsystem Requirements Trace
  - System Requirements Meeting Assessor Notes
- **Contents**
  - Summary of Significant Observations and Findings
    - requirements driving the subsystem architecture
    - representative quality cases (e.g., claims, arguments, evidence)

- Recommendations (e.g., for better engineering the architecture-significant subsystem requirements, for producing better architecture quality cases, or for improving/tailoring the system architecture assessment method)
- Requests for Further Information

6.2.8 Subsystem Requirements Review Meeting Minutes

- **Definition -** the official record of the proceedings of an individual subsystem requirements meeting
- **Objectives**
  - Record the significant observations and findings made during the subsystem requirements meeting about
    - requirements that drive the subsystem architecture
    - representative partial samples of architects' cases (e.g., claims, arguments, evidence) presented during the meeting
  - Record any significant recommendations such as recommendations for
    - better engineering the architecture-significant subsystem requirements
    - producing better architecture quality cases
    - improving/tailoring the system architecture assessment method
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the meeting attendees (prior to official publication)
  - Maintained by the assessment team (factual corrections only)
  - Used by the
    - assessment team to document their findings and recommendations and to act as an input for updating/tailoring the architecture assessment procedure and subsystem requirements checklist
    - management team to understand the status and quality of the requirements that drive the architecture
    - subsystem architecture team to understand assessment team's observations and recommendations
    - subsystem requirements team to help improve the quality of their requirements that drive the subsystem architecture
    - top-level architecture team to help them understand the status of the requirements driving the subsystem architecture and the ability of the subsystem architecture team to defend the quality of their architecture during the subsystem architecture assessment meeting

- **Inputs**
  - Subsystem Requirements Meeting Checklist
  - Subsystem Requirements Meeting Preparatory Materials
  - Subsystem Requirements Meeting Presentation Materials
  - Subsystem Requirements Trace
  - System Requirements Meeting Assessor Notes
  - Subsystem Requirements Meeting Outbrief
- **Contents**
  - Meeting Date and Location
  - Attendees
    - members of the assessment team
    - members of the subsystem architecture team
    - members of the subsystem architecture team
  - Subsystem Requirements
    - significant observations
    - subsystem architecture quality cases
    - significant observations
  - Recommendations for Improvement
    - requirements
    - architecture quality cases
    - architecture assessment method
  - Action Items (by reference – see Section 6.1.7)

6.3 Subsystem Architecture Assessment Work Products

As illustrated in Figure 18, the following work products are produced for or during the individual subsystem architecture assessment meetings:

1. Subsystem Architecture Assessment Checklist
2. Subsystem Architecture Assessment Preparatory Materials
3. Subsystem Architecture Assessment Meeting Agenda
4. Subsystem Architecture Assessment Presentation Materials
5. Subsystem Architecture Assessment Assessor Notes
6. Subsystem Architecture Support Matrix
7. Subsystem Architecture Assessment Meeting Outbrief
8. Subsystem Architecture Assessment Meeting Report

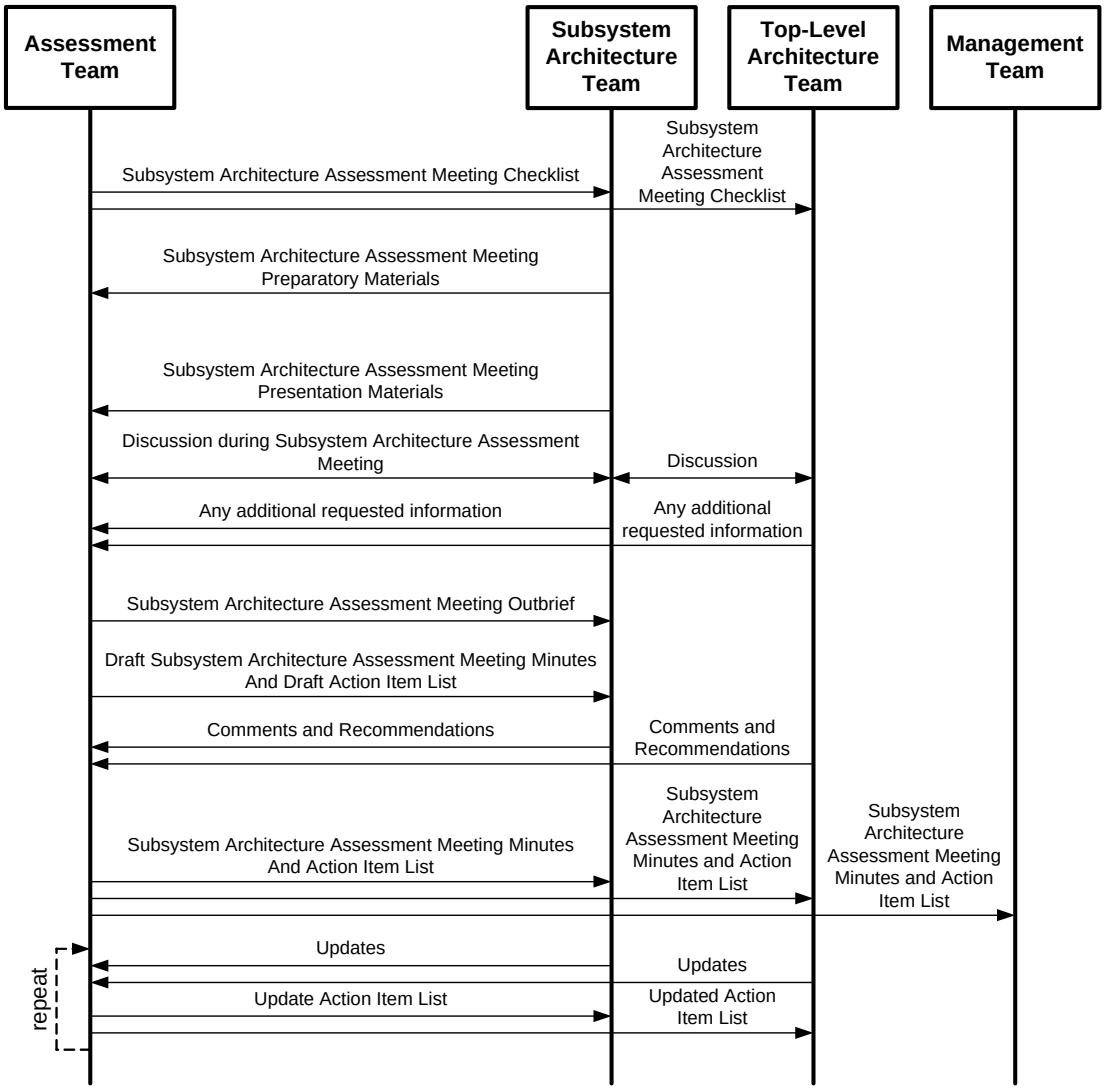

Figure 18: Subsystem Architecture Assessment Work Product Flow

The subsystem architecture assessment meeting work products possess the relationships depicted in Figure 19.

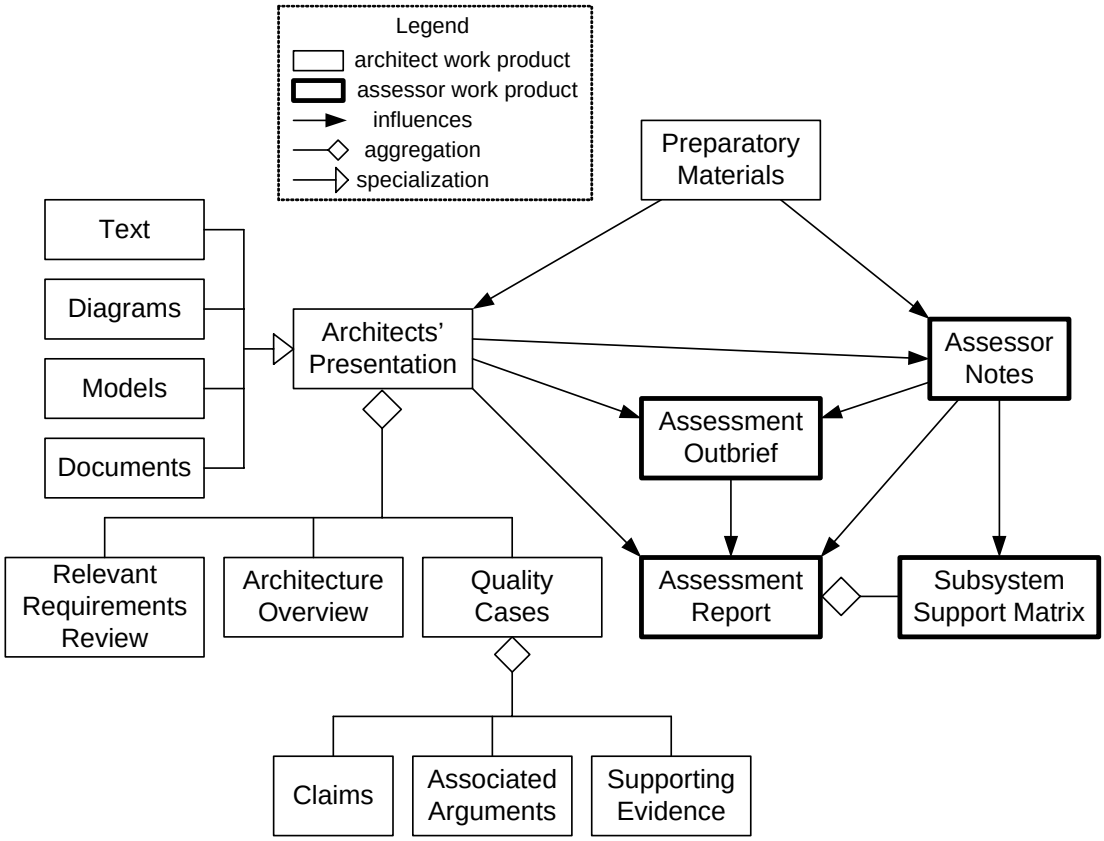

Figure 19: Subsystem Architecture Assessment Work Product Relationships

6.3.1 Subsystem Architecture Assessment Checklist

- **Definition -** a checklist used by the assessment team during the subsystem architecture assessment meetings to help them identify defects
- **Objectives**
  - Help the subsystem architecture team better prepare for the subsystem architecture assessment meeting.
  - Help the assessment team identify defects associated with the subsystem architecture team's quality cases.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the subsystem architecture team
  - Maintained by the assessment team
  - Used by the
    - assessment team to identify defects associated with the subsystem architecture team's quality cases
    - subsystem architecture team to understand what is expected of them during the relevant subsystem architecture assessment meeting

- **Inputs**
  - Architecture Assessment Method (tailored)
  - Default Subsystem Architecture Assessment Meeting Checklist (example in Appendix D of this handbook)
  - Initial Kickoff Meeting Minutes
- **Contents**
  - questions related to potential defects associated with the subsystem architecture overview
  - questions related to potential defects associated with the quality cases (e.g., claims, arguments, evidence)

6.3.2 Subsystem Architecture Assessment Preparatory Materials

- **Definition** - preparatory materials that the architects provide to the assessment team before a subsystem architecture assessment meeting, allowing plenty of time for review
- **Objectives**
  - Enable the assessment team to properly prepare for the assessment meeting.
  - Maximize the effectiveness and efficiency of the assessment meeting.
- **Stakeholders**
  - Produced by the subsystem architecture team
  - Reviewed by the
    - architecture team (prior to delivery to the assessment team)
    - assessment team (prior to the assessment meeting)
  - Maintained by the subsystem architecture team
  - Used by the assessment team to properly prepare for participation in the subsystem architecture assessment meeting
- **Inputs**
  - architectural diagrams, models, and documentation
  - architects' knowledge of the architecture
- **Contents**
  - a brief overview of the subsystem's architecture (e.g., training materials)
  - any significant updates to the architecturally significant requirements since the requirements meeting
  - quality cases (by quality factor):
    - architects' claims (quality goals and requirements)
    - architects' arguments (architectural decisions and rationales)
    - evidence in the form of official architecture diagrams, models, or documents showing the architecture's support for its associated architecturally significant requirements (either with relevant parts highlighted or an index identifying relevant parts)

6.3.3 Subsystem Architecture Assessment Presentation Materials

- **Definition** - presentation materials that the architects present during the subsystem architecture assessment meeting
- **Objectives**
  - Communicate the architect's case that their architecture adequately fulfills the derived and allocated architecturally significant requirements.
  - Enable the assessment team to assess the architecture against its architecturally significant requirements.
- **Stakeholders**
  - Produced by the subsystem architecture team
  - Reviewed by the top-level architecture team (prior to presentation)
  - Maintained by the subsystem architecture team
  - Used by the
    - assessment team to understand the subsystem architecture and the architecture team's quality cases
    - subsystem architecture team to make their quality cases that their subsystem architecture sufficiently supports its derived and allocated architecturally significant requirements
- **Inputs**
  - architectural diagrams, models, and documentation
  - architects' knowledge of the architecture
  - subsystem architecture assessment meeting preparatory material (e.g., evidence)
- **Contents**
  - brief overview of the subsystem's architecture
  - for each type of architecturally significant requirement (typically, quality attribute)
    - claims that the subsystem architecture adequately supports its derived and allocated architecturally significant requirements
    - clear and compelling arguments justifying these claims in terms of the architectural decisions made and their rationales
    - summaries and typical examples of the evidence supporting these arguments
  - Answers to questions posed by members of the assessment team

6.3.4 Subsystem Architecture Assessment Meeting Agenda

- **Definition** - the informal agenda for a single subsystem architecture assessment meeting
- **Objective**

  Inform meeting attendees of meeting topics and associated times.

- **Stakeholders**
  - Produced by the
    - assessment team
    - subsystem architecture team
  - **Not** reviewed
  - **Not** maintained
  - Used by meeting attendees
- **Inputs**
  - Architecture Assessment Procedure
  - discussions
- **Contents**
  - Meeting Topics and Times including
    - Introductions and Meeting Logistics
    - Quality Factors
    - Quality Cases (by quality factor), including discussions and questions
    - Schedule of Coming Events
    - Meeting Wrap-Up

6.3.5 Subsystem Architecture Assessment Meeting Assessor Notes

- **Definition** - informal notes that an individual assessor takes during the subsystem architecture assessment meeting
- **Objectives**
  - Capture key findings.
  - Capture significant observations.
  - Capture recommendations.
  - Capture personal action items.
- **Stakeholders**
  - Produced by individual members of the assessment team who attend the subsystem architecture assessment meeting
  - **Not** reviewed
  - **Not** maintained
  - Used by the scribe of the assessment team, to use as input to the subsystem architecture assessment meeting outbrief and meeting minutes
- **Inputs**
  - Subsystem Architecture Assessment Meeting Checklist
  - Subsystem Architecture Assessment Meeting Preparatory Materials
  - Subsystem Architecture Assessment Meeting Presentation Materials
  - architects' answers to questions asked by members of the assessment team

- **Contents**
  - For each type of architecturally significant requirement (typically quality attribute), informal notes capturing any observations or key findings
  - Recommendations regarding improving the
    - architects' quality cases
    - architecture assessment method
  - questions asked and answers given
  - any other information that the individual assessors consider significant and worthy of writing down

6.3.6 Subsystem Architecture Support Matrix

- **Definition** – a matrix that documents the architecture's level of support for the different types of architecturally significant requirements in terms of its subsystems' support

  The subsystem support matrix is developed after the architects have presented their cases as part of the preparation for producing the subsystem architecture assessment meeting outbrief and report.

- **Objective**

  Communicate the assessment that each subsystem received for its support for each type of architecturally significant requirement.

- **Stakeholders**
  - Produced by the assessment team (typically the team leader)
  - Reviewed by the assessment team
  - Maintained by the assessment team
  - Used by assessment team, to summarize the results of the assessment of the subsystem architecture

- **Inputs**
  - Subsystem Architecture Assessment Meeting Preparatory Materials
  - Subsystem Architecture Assessment Meeting Presentation Materials
  - architects' answers to questions asked by members of the assessment team
  - Subsystem Architecture Assessment Meeting Assessor Notes

- **Contents**
  - Rows representing sub-subsystems
  - Columns representing types of architecturally significant requirements (typically quality factors)
  - Cells containing color-coded assessment ratings
    - **Green** – Architecture adequately supports achievement of all requirements.

      The architecture team has presented clear and convincing arguments backed up by sufficient underlying evidence to persuade the assessment team that the architecture adequately supports the systems' achievement of all of its allocated and derived quality requirements.

– **Yellow** – Architecture may or may not adequately support achievement of all requirements.

  The architecture team has either **not** presented adequate arguments or **not** provided adequate underlying evidence to completely persuade the assessment team that the architecture adequately supports the systems' achievement of all of its allocated and derived quality requirements.

– **Orange** – Architecture does *not* adequately support achievement of all requirements.

  The architecture team has **not** presented enough clear and convincing arguments backed up by sufficient underlying evidence to persuade the assessment team that the architecture adequately supports the systems' achievement of all of its allocated and derived quality requirements

– **Red** – Architecture makes it difficult but not impossible to achieve some requirements.

  The architecture team has presented some arguments or provided some evidence to convince the assessment team that the architecture makes if difficult (but not impossible) for the system to achieve all of its allocated and derived quality requirements.

– **Black** – Architecture prevents achievement of some requirements.

  The architecture team has presented some arguments or provided some evidence that has given the assessment team significant reasons to believe that certain aspects of the architecture will prevent the system from achieving some of its allocated and derived quality requirements.

– **N/A** – The associated type of architecturally significant requirements is not applicable to the associated sub-subsystem.

– **TBD** – Indicates that the sub-subsystem's architecture is not ready for assessment.

Assessments	Capacity	Interoperability	Performance	Reliability	Safety
Subsystem 1	Green	Green	Green	Yellow	Green
Subsystem 2	Green	Yellow	Red	Green	Green
Subsystem 3	Green	N/A	Yellow	Green	Yellow
Subsystem 4	TBD	TBD	TBD	TBD	TBD
…	…	…	…	…	…
Subsystem N	Green	Green	Orange	Green	Green

Table 1: Example Subsystem Support Matrix

6.3.7 Subsystem Assessment Meeting Outbrief

- **Definition** - an interim top-level summary of the results of a subsystem architecture assessment meeting

- **Objectives**
  - Communicate a summary of the assessment team's interim results from the subsystem architecture assessment meeting.
  - Elicit comments and recommendations from the subsystem architecture team, especially to correct any factual misunderstandings before they are incorporated into the meeting's minutes.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the assessment team (internally before the draft is sent to the subsystem architecture team)
  - Temporary document that is not maintained, but rather superseded by the subsystem architecture assessment meeting report
  - Used by the
    - assessment team to communicate their interim results to the subsystem architecture team
    - subsystem architecture team to understand the assessment team's interim observations, findings, and recommendations related to the architecture team's presentation
- **Inputs**
  - Subsystem Architecture Assessment Meeting Preparatory Materials
  - Subsystem Architecture Assessment Meeting Presentation Materials
  - Subsystem Architecture Assessment Meeting Assessor Notes
  - Subsystem Architecture Support Matrix
  - architects' answers to questions asked by members of the assessment team
- **Contents**
  - Assessment grades for the subsystem architecture
    - grade for overall subsystem architecture support for derived architecturally significant types of requirements allocated to it
    - grades for each sub-subsystem for each type of architecturally significant requirements (subsystem support matrix)
  - Key Findings of the Subsystem Architecture Assessment
  - Key Recommendations regarding improving the
    - architects' quality cases
    - architecture assessment method

6.3.8 Subsystem Architecture Assessment Meeting Report

- **Definition** - a report that documents the results of the subsystem architecture assessment meeting

  This report is developed and distributed to its stakeholders within two to four weeks after the assessment meeting.

- **Objectives**
  - Document the degree to which the subsystem architecture being assessed supports the derived architecturally significant requirements allocated to it.
  - Communicate this information and other results of the subsystem architecture assessment meeting to all stakeholders.
- **Stakeholders**
  - Produced by the assessment team (primarily a small subteam including the scribe)
  - Reviewed by the assessment team (internally before the draft is sent to the subsystem architecture team)
  - Maintained by the assessment team (primarily for correction of factual errors)
  - Used by the
    - assessment team to communicate their final results to the subsystem architecture team and to update the architecture assessment procedure
    - subsystem architecture team to understand the assessment team's key findings, general observations, and recommendations related to the subsystem architecture
- **Inputs**
  - Subsystem Architecture Assessment Meeting Preparatory Materials
  - Subsystem Architecture Assessment Meeting Presentation Materials
  - Subsystem Architecture Assessment Meeting Assessor Notes
  - Subsystem Architecture Support Matrix
  - Subsystem Architecture Assessment Meeting Outbrief
  - architects' answers to questions asked by members of the assessment team
- **Contents**
  - Executive Overview, which is a top-level summary of the results of the subsystem architecture assessment
  - Introduction, which is an overview of the subsystem architecture assessment:
    - Assessment Objectives
    - Assessment Scope
    - Assessment Participants
  - Key Findings
  - Major Recommendations
  - Lessons Learned (concerning the assessment method)
  - Appendices: Acronym List

6.4 System Architecture Quality Assessment Summary Work Products

The following work products are produced for or during the system architecture quality assessment summary meeting:

1. System Summary Subsystem Matrix
2. System Summary Meeting Presentation Materials

3. System Architecture Assessment Summary Meeting Agenda
4. System Architecture Assessment Summary Meeting Assessor Notes
5. System Architecture Quality Assessment Summary Report

6.4.1 System Summary Subsystem Matrix

- **Definition** - the matrix that summarizes the results of all of the subsystem assessments
- **Objective**

  Provide a concise overview of the assessment results.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the assessment team
  - Maintained by the assessment team (primarily for correction of factual errors)
  - Used by the
    - assessment team to provide input to the system summary meeting presentation material and the system architecture quality assessment summary report
    - management team to get subsystem-specific architecture quality information to identify architectural risks and to determine where to allocate resources
    - top-level architecture team to get subsystem-specific architecture quality information to identify architectural risks and to determine where to allocate resources
- **Inputs**
  - Subsystem Architecture Assessment Reports
  - Subsystem Architecture Support Matrices
- **Contents**
  - Collection of Subsystem Architecture Support Matrices

6.4.2 System Summary Meeting Presentation Materials

- **Definition** - the preparatory materials for the summary system architecture assessment meeting
- **Objective**

  Allow invitees of the summary system architecture assessment meeting to prepare for the meeting by seeing an initial version of the materials to be presented during the meeting.
- **Stakeholders**
  - Produced by the assessment team
  - Reviewed by the architecture team (prior to delivery to the meeting attendees)
  - Maintained by the assessment team (primarily for correction of factual errors)
  - Used by the
    - assessment team to provide input to the system summary meeting presentation material and the system architecture quality assessment summary report

  - management team to get subsystem-specific architecture quality information to identify architectural risks and to determine where to allocate resources
  - top-level architecture team to get subsystem-specific architecture quality information to identify architectural risks and to determine where to allocate resources

- **Inputs**
  - Subsystem Architecture Assessment Reports
  - Subsystem Architecture Support Matrices
  - System Summary Subsystem Matrix
- **Contents**
  - Restatement of Assessment Objectives
  - Summary of Assessment Method including approaches used to summarize the qualities of the subsystem architectures
  - Summary of the Quality of the Architectures of the Subsystems
  - Summary of the Quality of the System Architecture
  - Initial Lessons Learned

6.4.3 System Architecture Assessment Summary Meeting Agenda

- **Definition -** the informal agenda for the summary system architecture assessment meeting
- **Objective**

  Inform meeting attendees of meeting topics and associated times.
- **Stakeholders**
  - Produced by the assessment team
  - **Not** reviewed
  - **Not** maintained
  - Used by the meeting attendees
- **Inputs**
  - Architecture Assessment Procedure
  - discussions
- **Contents**
  - Meeting Topics and Times including
    - Statement of Assessment Objectives
    - Overview of Assessment Method
    - Summary of the Quality of the System Architecture in terms of the quality of the architectures of the subsystems
    - Final Lessons Learned
    - Meeting Wrap-Up

6.4.4 System Architecture Assessment Summary Meeting Assessor Notes

- **Definition** – informal notes that an individual assessor takes during summary system architecture assessment meeting
- **Objectives**
  - Capture observations and recommendations made by the meeting attendees for improving the
    - final System Architecture Quality Assessment Summary Report
    - architecture assessment procedure
    - architecture assessment training materials
    - account of personal action items
- **Stakeholders**
  - Produced by individual members of the assessment team that attend the system architecture assessment summary meeting
  - **Not** reviewed
  - **Not** maintained
  - Used by the scribe of the assessment team as input to the final System Architecture Quality Assessment Summary Report
- **Inputs**
  - observations and recommendations made by the meeting attendees
- **Contents**
  - Observations and Recommendations for improving the
    - System Architecture Quality Assessment Summary Report
    - Architecture Assessment Procedure
    - Architecture Assessment Training Materials
  - Any other information that the individual assessor considered significant and worthy of writing down

6.4.5 System Architecture Quality Assessment Summary Report

- **Definition** – the report that documents the overall quality of the system architecture
- **Objectives**
  - Document the quality of the system architecture in terms of the quality of the architectures of its subsystems.
  - Summarize the
    - assessment objectives
    - system architecture quality assessment method used
    - assessment lessons learned
  - Produced by the assessment team

  - Reviewed by the
    - architecture team (prior to delivery to the assessment team)
    - assessment team (prior to the assessment meeting)

- **Inputs**
  - Subsystem Architecture Assessment Reports
  - Subsystem Architecture Support Matrices
  - System Summary Subsystem Matrix
  - System Summary Presentation Materials
  - Discussions and inputs from the assessment team and stakeholders including the system and subsystem requirements and architecture teams
- **Contents**
  - Statement of Assessment Objectives
  - Overview of Assessment Method
  - Summary of the Quality of the System Architecture in terms of the quality of the architectures of the subsystems
  - Final Lessons Learned

7 QUASAR Lessons Learned

During the development and initial use of the successive versions of this system architecture quality assessment method, the lessons documented in this section were learned and have been incorporated into the current version of the method.

7.1 System Architecture Assessment Initiation Phase

The following lessons are related to the System Architecture Assessment Initiation Phase of the QUASAR method:

1. **Ensure adequate assessment team membership.**

   Lesson: If practical, ensure that the assessment team includes at least one member who is familiar with the QUASAR method and quality cases. Also ensure that the assessment team includes one or more members who are familiar with the domain of the system (or subsystems) being assessed. Finally, ensure that the assessment team includes one or more members who are familiar with the system and if possible, its architecture.

   Rationale:

   a. Independent architecture quality assessments are relatively new.
   b. It is difficult to successfully implement an assessment method without access to at least one person with prior experience.
   c. Much of a system's architecture is greatly impacted by the type of system being architected. For example, there is a great difference between hard real-time, safety-critical embedded systems and Web-based financial systems.
   d. Having at least one member who is familiar with the system and its architecture makes it easier to determine what information to ask for if the architects are unclear as to what makes good evidence.

2. **Provide architecture assessment training early.**

   Lesson: The assessment team should provide an initial training session for members of both the assessment team(s) and the top-level architecture team on the objectives, ground rules, and method (e.g., tasks, techniques, and work products) for performing the architecture quality assessment. This training should be of short duration. The timing of this training should be appropriate. The training should concentrate on the generalization of safety case concepts as applied to system architecture quality assessment: the architects' *claims* that their architecture has sufficient quality, their clear and compelling *ar-*

guments supporting these claims, and legitimate *evidence* justifying belief in these arguments.

Rationale:

a. The assessment method will probably be new to the architects and most members of the assessment team.
b. The training helps ensure that the members of these teams will
   – understand their responsibilities and how to perform their tasks
   – know what information is basic information, what information is legitimate and appropriate as evidence supporting the architects' arguments, and how to differentiate the two
   – make fewer mistakes when applying the method
   – be more efficient in achieving the goals of the assessments
c. The top-level architecture team typically represents all of the lower architecture teams during
   – tailoring of the method for use on a series of assessments
   – accepting of (i.e., committing to fulfill) their responsibilities under the assessment method

3. Provide early architecture development process training.

Lesson: The top-level architecture team should provide an initial training session on the architecture development and maintenance process including the following:

a. architecting tasks, techniques, and conventions (standards, procedures, guidelines)
b. architecture roles and responsibilities
c. architecture work products including types (e.g., models, views, documents), purposes, conventions, etc.
d. The architects should clearly show how the architecture will evolve in an incremental development cycle and how the architecture varies across variants within a product line. This should include how they identify increments and variants within requirements repositories, requirements and architecture models, and requirements and architecture documents. This is difficult to do with today's immature tools.

Rationale:

a. This training is quite valuable to members of the assessment team in terms of setting their expectations.
b. This training helps members of the assessment team determine if the set of the architects' work products (especially views) is both adequately complete and also appropriate (e.g., not confusing requirements models such as mistaking use case models for architecture models).
c. If performed sufficiently early in the development cycle, providing this training can influence the
   – requirements engineers' method, thereby enabling them to properly engineer the architecturally significant requirements that drive the architecture
   – architects' method for developing the architecture, thereby enabling them to
      o produce the arguments and evidence needed for the assessment as a natural part of their architecture method

- avoid having to develop significant documentation for the assessments
- decrease the impact of the assessments on their busy schedules

– A large system is too big and complex to be developed using the waterfall development cycle. The use of an incremental, iterative, and parallel development cycle means that different parts of the architecture will reach different levels of maturity at different times.

4. Set architecture assessment objectives.

Lesson: During the initial kickoff meeting, the assessment team(s) and top-level architecture team should develop a consensus regarding the objectives of the set of assessments of the system's overall architecture.

Rationale:

a. Architecture assessments can have different objectives.
b. These objectives influence the importance and level of detail of the assessments.
c. A consensus on the assessment objectives enables the teams to
   – agree on the proper tailoring of the assessment method
   – set the overall assessment scope

5. Set overall assessment scope.

Lesson: It is important to set the scope of the set of assessments of the overall system architecture (or that subset of this architecture that is intended to be assessed). During the initial kickoff meeting, the top-level architecture team and the assessment team should agree on the scope of the overall assessment in terms of the intended:

a. specific architectural elements (e.g., subsystems) to be assessed
b. level of detail within (i.e., average/maximum depths in the hierarchical architecture below) these elements to which the architecture will be assessed (e.g., subsystems and sub-subsystems)
c. default, specific types of architecturally significant requirements (e.g., quality factors) against which to evaluate these architectural elements

Rationale:

a. This knowledge is needed to estimate the total resources and costs to be allocated to the assessment.
b. This knowledge is needed to develop an initial general schedule for the assessments to ensure that the architectural elements being assessed are sufficiently complete or important to justify the cost and effort needed to perform a proper assessment.
c. Setting the overall scope helps to ensure that the system will satisfy the customer's expectations and meet the customer's needs.
d. Subsystems of the subsystems to be assessed may
   – be at the level of design, rather than architecture and thus be out of scope
   – not be sufficiently complete or important to justify the cost and effort needed to perform a proper assessment

6. Agree on definitions.

Lesson: During the requirements meeting, it is critical for the architecture and assessment teams to agree on the exact definitions of the quality factors (and their subfactors)

to be used as the basis for the architecture assessment. It is also important to agree on what the corresponding quality requirements are. Although the importance of different quality factors and quality requirements often differs from subsystem to subsystem, different subsystems should not have different definitions or "interpretations" of the quality factors.

Rationale:

a. In practice, there is often confusion between the definitions of quality factors and the definition of the associated quality factor requirements.
b. Although international standards defining quality factors exist, different people and organizations typically have different definitions of quality factors in practice.
c. These definitions may legitimately vary within large systems because of the application domains of different architectural elements.
d. The use of different definitions leads of confusion, miscommunication, and the waste of time and effort. For example, some people may not understand the differences between availability, reliability, and stability. Others may confuse the stability of a *system* in terms of system failure with the stability of the system's *architecture* in terms of resistance to change in spite of requirements change.

7. Primarily assess by subsystems within tiers.

Lesson: Assessments should be performed on individual architectural elements within the architecture's hierarchical decomposition tier structure (e.g., system of systems, systems, subsystems, sub-subsystems, etc.).

Rationale:

a. Individual architectural elements (e.g., subsystems) tend to provide a reasonable size for assessments in terms of limiting the duration of individual assessments.
b. Clarifying the tier levels make it easier to differentiate architecture (tier n) versus detailed design (tier n+1) as well as requirements (from tier n-1).
c. It would be confusing to try to assess the architecture across too many tier levels.

Exceptions:

a. Sometimes, multiple subsystems are highly related in terms of architectural components, mechanisms used to support the architecturally significant requirements, and architects. Therefore, it *may* be possible to assess two architectural elements during a single week.
b. It is often important to perform a small number of "deep-dives" to verify actual architectural support via a subsystem's sub-subsystems.

8. Ensure adequate resources and planning.

Lesson: It is important that all teams include assessments in their team plans and team schedules so that they can ensure that their members are available and have adequate funding to support the assessments.

7.2 Subsystem Requirements Review Phase

The following lessons are related to the Subsystem Requirements Review Phase of the QUASAR method:

1. **Hold requirements review prior to architecture assessment.**

   Lesson: For each architectural element being assessed, the relevant development team(s) (i.e., requirements team and/or architecture team) should perform an initial requirements review for the assessment team. This review should be held a sufficient amount of time prior to the architecture assessment.

   Rationale:

   a. Such a review helps ensure the existence of the architecturally significant quality goals and requirements.
   b. It helps ensure a common understanding of these goals and requirements.
   c. It minimizes confusion over and misinterpretation of the requirements and use cases.
   d. Listing the driver requirements after listing the architectural decisions is of little value, especially if the requirements are merely identified by number. However, even if the requirements are presented and agreed upon during an initial requirements/driver meeting, it helps to briefly list them prior to describing how the architecture supports the quality factors so as to provide a clear rationale/justification for the architecture decisions.

2. **Identify architecturally relevant requirements.**

   Lessons: This is not a requirements assessment, but rather an architecture assessment. Cover the requirements only to the extent that the allocation of requirements to major architectural elements needs to be understood. Concentrate on the architecturally significant requirements so that the architectural support for these requirements can be assessed.

   Rationale:

   a. It is very difficult to determine if the architecture sufficiently supports a quality factor without well-specified requirements specifying a required minimum measurable level of quality. For example, just how scalable must the element be and in what way? Without knowing how good the architecture must be, it is very difficult to objectively determine if the architecture is good enough.

3. **Concentrate on quality requirements.**

   Lesson: During the requirements meeting, it is important to ensure that the architecture and assessment teams understand the differences between quality factors and their associated quality requirements that have (or should have) been derived and allocated to the subsystem being assessed. It is also important to accept these quality requirements as requirements and not merely as “assessment criteria.”

   Rationale:

   a. The definition of a quality factor should not change from subsystem to subsystem.
   b. A quality factor is not a requirement, but merely a characteristic of the system or subsystem that may or may not be there. It does not in and of itself imply any impact on the subsystem unless there is a requirement for the subsystem to exhibit that quality factor to a specific degree.

c. Although there is a cost associated with requirements because of the associated verification effort, calling quality requirements "assessment criteria" minimizes their influence on the architecture.

d. Lack of agreement on the relevant quality requirements allocated to the architecture means that these requirements do not end up being derived and allocated to the architecture. This makes it difficult to assess the architecture against these non-derived, non-allocated requirements.

4. Set individual assessment scope.

Lesson: It is important to set the scope of each assessment. At the requirements meeting prior to the assessment meeting, the architecture and assessment teams should agree on the scope of the assessment in terms of the

a. specific architectural elements (e.g., subsystems) to be assessed

b. level of detail within (i.e., depth in the hierarchical architecture below) these elements against which the architecture will be assessed (e.g., subsystems and sub-subsystems)

c. specific types of architecturally significant requirements (e.g., quality factors) against which to evaluate these architectural elements

Rationale:

a. All of the subsystems may not be sufficiently complete or important to justify the cost and effort needed to perform a proper assessment. It makes little sense to evaluate a subsystem if its associated requirements are still largely volatile and unspecified.

b. Subsystems of the subsystems to be assessed
   - may be at the level of design rather than architecture and thus be out of scope
   - may not be sufficiently complete or important to justify the cost and effort needed to perform a proper assessment
   - have different quality factors, the relevance and importance of which may vary from subsystem to subsystem

c. Setting the scope of the assessment enables the
   - architecture team to minimize the amount of effort required to prepare for the assessment meeting by limiting the amount of preparatory documentation (e.g., presentation materials) they must develop for the assessment meeting
   - assessment team to minimize the amount of effort required to prepare for the assessment meeting by restricting their reading to relevant architectural documentation, diagrams, and models
   - assessment team to create an appropriate assessment notes template that is limited to the specific subsystems to be assessed

5. Select architecture-appropriate quality factors.

Lesson: Determine the architecturally significant quality factors against which to evaluate an architectural element during its associated requirements meeting. At the same time, determine the relative priorities of these selected quality factors.

Rationale:

a. Intraoperability can be as (or more) important as interoperability when a system integrator is integrating subsystems provided by multiple subcontractors or vendors.

b. Forcing all architecture assessments to be made against the same quality factors often forces the architects to stretch the meaning of a relatively inappropriate quality factor so that they can "get a check in the box."
c. Forcing architects to prepare for and undergo an assessment against an inappropriate quality factor gives the architects the impression that the assessment is irrelevant and largely a waste of time.
d. The most important quality factors in terms of their impact on the architecture vary from architectural element to architectural element based on its application area (e.g., aircraft vs. trainer).

6. Train subsystem architecture team.

Lesson: Break the assessment of a subsystem's architecture into two parts: an initial review to train the assessors on the architecture followed by multiple assessments to test portions of the architecture.

Rationale: It is good to give the architects (at an early meeting)

a. examples of expected arguments (i.e., architectural decisions)
b. examples of types of evidence (e.g., context diagram for interoperability)
c. checklists for verifying their readiness for the assessment meeting
d. template for presentations

7. Select an adequate set of relevant quality factors.

Lesson: Select an adequate set of quality factors that is appropriate for the subsystem being assessed. Ensure the development of a consensus on the quality factors among all teams (i.e., assessment, requirements, and architecture) that are stakeholders in a subsystem assessment. If necessary, resort to a final decision by an assessment authority (e.g., assessment team leader).

Rationale: Because of the assessment cost in terms of effort and availability of critical personnel, there is a strong tendency (especially within the architecture team) to limit the scope of the assessments in terms of the quality factors on which quality cases are developed, sometimes beyond that which is appropriate. Whereas the subsystem architecture team may feel that they are the only ones qualified to decide regarding their subsystem, other teams such as the requirements team, system architecture team, and assessment team (representing the acquisition and user organizations) are also major stakeholders in ensuring that important quality factors are not omitted.

Differences between subsystems provide important, legitimate reasons for choosing different quality factors on which to base quality cases. The quality factors often need to be expanded to include programmatic constraints such as budget (affordability) and schedule limitations.

Finally, there will be significant argumentation and loss of collaboration if a consensus is not developed early.

8. Ensure understanding of relevant quality factors.

Lesson: It is critical for all meeting attendees to understand the quality factors and the architectural support for each of them.

Rationale: Sometimes attendees become confused about the meanings of the different quality factors or what documentation constitutes valid evidence. This can lead to argument and evidence that does not actually support the associated requirements. For example, testability is the degree to which the architecture supports testing rather than how much testing one intends to perform. Plans to use a testing tool are also not evidence that the system *architecture* adequately supports testability.

9. **Develop requirements trace.**

   Lesson: The architects should create a trace from the requirements to the associated quality factors.

   Rationale: Traces are very useful to ensure that the architecture supports the associated quality-factor-related requirements. Preparing a requirements trace from the quality-relevant requirements to the individual quality factors helps the assessment team determine potential holes in the requirements and the relevant drivers of the architecture.

7.3 Subsystem Architecture Assessment Phase

The following lessons are related to the Subsystem Architecture Assessment Phase of the QUASAR method:

1. **Provide initial overview of subsystem architecture.**

   Lesson: The subsystem architecture team should start the subsystem architecture assessment meeting by presenting a top-level overview of the subsystem architecture to the assessment team.

   a. **Keep overview short.**

      Lesson: Keep the initial overview brief.

      Rationale: A *short* briefing providing an overview of the subsystem architecture will not take an excessive amount of time away from the architects' presentation of their quality cases. In fact, it will save time by ensuring that all assessors have the same minimal foundation in the subsystem, the architecture of which is being assessed.

   b. **Present primary diagrams.**

      Lesson: This overview should include the primary diagrams documenting the major architectural components of the subsystem and their major relationships (e.g., collaboration diagram, network diagram, and context diagram).

      Rationale: Because of scheduling conflicts, not all of the assessors (especially subject matter experts brought in specifically for the assessment) will have had the opportunity to adequately review the subsystem architecture prior to the meeting. Being primary evidence supporting the architects' arguments, these diagrams set the context for the quality cases that follow. These diagrams are important evidence that back up the architects' arguments by greatly clarifying some of the most important architectural decisions.

c. Present primary decisions.

Lesson: This initial overview could also include a brief listing of the major architectural policy decisions (e.g., architecture patterns and mechanisms) that are to be flowed down into these lower level architectural components (e.g., sub-subsystems and associated processes).

Rationale: This listing helps provide a good introduction to the architects' arguments in their quality cases.

d. Mount diagrams on walls.

Lesson: The subsystem architecture team should mount large copies of these primary architectural diagrams on the walls of the assessment meeting room.
Rationale: The assessors can easily refer to these diagrams during the later presentations and during the meeting breaks.

e. Highlight primary architectural decisions.

Lesson: If practical, the subsystem architecture team should clearly identify their major relevant architectural decisions on these wall-mounted diagrams (e.g., use highlighters to document relevant hardware redundancy for reliability cases or classified data flows for security cases).

Rationale: The entire diagram is not relevant to the arguments of specific quality cases. Marking the important parts of the diagrams makes it easier for the architects to convey their arguments to the assessors. Manual marking of the relevant quality case perspectives of the architectural views is appropriate because tool support to provide these perspectives is rare.

f. Leave diagrams up.

Lesson: These important diagrams should left up on the walls during the entire meeting. If the assessors are going to develop their outbrief and the initial parts of their meeting report in the same room where the assessment meeting it taking place, then these diagrams should remain on the walls until the assessment team leaves.

Rationale: The assessors need to refer to these diagrams often, and the large size of these diagrams makes it easy for multiple assessors to gather around them and discuss the architecture and the architects' decisions.

2. Focus on assessing the existing architecture.

Lessons: It is important to ensure that the assessment team stays on topic so that the architectural assessment does not devolve into an assessment of the requirements, design, or implementation. Although occasional deep dives into the design may be used as a way to validate the quality of the architecture in terms of its implementability, care must be taken to ensure that excessive time is not wasted going down an interesting but relatively unimportant rabbit hole.

Rationale:

a. Technical members of the assessment team have a natural inclination and curiosity that can lead the assessment off track and out of scope.

b. Members of the architecture team are typically very busy and do not have unlimited time to allocate to an architecture assessment. They may legitimately resent what they might consider to be a waste of valuable time. They may also fear receiving a negative assessment on something (e.g., design or implementation) that is not yet ready for assessment and for which they have not adequately prepared.

3. Avoid a "trust me" approach.

Lesson: It is important that the architects understand that they must prove their case by providing an adequate amount of actual evidence to back up their arguments (i.e., show that they have in fact made the architectural decisions they say they have and show that these decisions were sufficient to ensure that the architecture adequately supports its derived and allocated requirements).

Rationale: Not relying solely on the architect's verbal description is not an accusation of lying but rather merely due diligence on the part of the assessment team.

4. Determine if arguments and evidence are incomplete.

Lesson: Members of the assessment team may discover that the architects have inadvertently not provided all of the arguments and evidence that they could have. For example, if the architects claim that their architecture adequately supports testability in terms of testing support for fault tolerance, they may only think of how the architecture enables them to insert failures, whereas other properties of the architecture (e.g., levels of modularity and cohesion, well-defined interfaces, publish-subscribe, and test ports) may also make it easier or more difficult to perform subsystem (acceptance), integration, and system (functional, fault tolerance) testing.

Rationale: Quality cases are often incomplete due to missing arguments and evidence because of unfamiliarity with quality cases and exactly what is considered to be appropriate argument and evidence. Due to their greater familiarity with the QUASAR method, the assessment team (especially the subsystem liaison) can often identify missing parts of quality cases.

5. Use best quality case presentation order.

There are two main ways that the presentation of the architects' cases that their architecture supports its derived and allocated architecture-related requirements:

a. **By subsystem,** then by requirements type within subsystem.
b. **By requirements type**, then by subsystem support within requirements type.

Lesson: The best approach is to organize the presentation first by requirements type (e.g., quality factor) and then by subsystem within the subsystem, the architecture of which is being assessed.

Rationale: Assessing the architecture against the quality attributes is much more successful when performed first by quality attribute followed by subsystem support for the specific quality attribute as compared to the subsystem first followed by quality attribute within subsystem. The quality attribute first approach forces the architects to better address the individual quality attributes and to not rely on presenting preexisting general subsystem documentation, much of which is not rele-

vant to the scope of the assessment. It also makes the assessors' job of producing a report sorted by quality attribute much easier.

6. **Present architecture engineering tradeoffs.**

   Lesson: It is important for the architects to address the engineering tradeoffs they make between the different quality factors and the engineering tradeoffs they make between subsystems (e.g., allocating limited resources utilized by different subsystems such as timing budgets).

   Rationale: Quality factors are often somewhat incompatible in the sense that it is difficult to increase two simultaneously. For example, some of the techniques used to increase performance (local optimization) may decrease maintainability (e.g., local optimization for the sake of improving performance may increase coupling and decrease modularity, thereby decreasing maintainability). Similarly, increasing security may decrease usability (e.g., by requiring more frequent identification and authentication). Similarly, the system architecture not only consists of the architecture of the individual subsystems, it also addresses how these subsystems collaborate to meet the requirements of the overall system. Therefore, it is not sufficient to concentrate on individual quality factors and individual subsystems.

7. **Provide only some scenarios as evidence.**

   Lesson: All evidence should not be given in the form of scenarios.

   Rationale: While using single scenarios is a way of making quality requirements more meaningful, they often do not address all of the subsystems, thereby providing incomplete arguments and evidence than what is needed.

8. **Present structure before functional views.**

   Lesson: The architects should present the structural architecture in terms of element names, responsibilities, and relationships before they describe the functional behavior of how they collaborate to fulfill their assigned use cases. The structure needs to address the hardware structure, software structure, and deployment of software to hardware. The architects should present an adequately complete set of appropriate views, ensuring that no single view dominates the architecture documentation.

   Rationale: By emphasizing the components of the architecture and how they are related, it makes it easier to understand how these components collaborate to meet their allocated requirements. Otherwise, the architects may instead concentrate on a "logical architecture" of functions that the system must perform, rather than the actual architecture that helps implement these collections of functional *requirements*. Also, emphasizing functional views of the system may underemphasize the system's required qualities and how the system's architecture helps the system provide them.

9. **Keep evidence within scope.**

   Lesson: Evidence presented during the architecture assessment meeting should remain within the scope of the assessment:

a. Evidence presented by architects

Discussion: In practice, architects may not understand what kinds of documents make legitimate evidence to back up their arguments. Sometimes in their haste, the architects provide all of the documentation they have produced related to architecting (such as planning or procedural documentation), regardless of whether it provides evidence concerning the architecture's actual existing support for helping the subsystem achieve its derived and allocated quality requirements.

Lesson: As part of their quality cases, the architects should take care to only present legitimate evidence that supports their arguments (i.e., is within the scope of the assessment).

Rationale: Providing irrelevant material only wastes the time they took to provide it and the time the assessors took to review it. The assessors only have limited time during which to prepare for the assessment and reading irrelevant material prevents them from reviewing the relevant evidence. It also tends to give them the (potentially wrong) impression that the architects either do not have adequate evidence to back up their arguments or do not understand the documents they have.

b. Evidence requested by assessors

Discussion: During the assessors' questioning of the architects, the assessors may become aware of additional evidence that was not presented or provided to them. For example, an architect may raise a new argument when questioned about an aspect of the architecture, and the assessors may then legitimately ask the architect for supporting evidence. However, it is not uncommon for technical assessors to find quite interesting certain information mentioned in passing even though it does not directly support the architects' arguments.

Lesson: To the extent appropriate and practical, the assessors should carefully limit their requests for additional evidence to evidence that is actually relevant to the architects' arguments concerning the architecture's actual support for the subsystem achieving its quality goals and meeting its quality requirements.

Rationale: Due to their busy schedule and the limited time they have available to support the assessment, the architects should legitimately point out that the requested documentation (e.g., managerial, process, detailed design, or implementation documentation) is outside the scope of the assessment and need not be provided.

Counterargument: Because there are rarely clear, verifiable derived quality requirements to drive the architecture, the assessors sometimes must use process documentation to see what the contractor is using to drive their architecture (e.g., the reliability process used to determine the probability of software failure to determine how reliable the architectural components must be). Similarly, the assessors typically should report any out-of-scope defects they may serendipitously find.

10. **Ensure availability of actual architects.**

    Lesson: It is very useful to have the actual subsystem architects available during the assessment meeting. Trying to save the subsystem architects time by having only the primary system chief architect present their quality cases for them is not an effective way to run the assessment meeting.

    Rationale: Having the subsystem architects present their own quality cases is best because the subsystem architects

    a. understand their own architectures better then the chief architect does
    b. are immediately available to answer detailed questions posed by the assessment team
    c. can quickly locate and display the specific relevant information within the large amount of evidentiary documentation

11. **Use existing documentary evidence.**

    Lesson: To the extent practical, the architects should present their quality cases using existing architectural documentation.

    Rationale: Architects should have documented their architectural decisions and associated rationales as a normal part of their architecture process. By including this existing official (e.g., under configuration control) documentary evidence in their quality cases, architects both minimize the work needed to prepare for an assessment and ensure that the actual architecture is presented.

    Unfortunately, some architects either may not do an adequate job of documenting their architecture in the first place or else they may not keep their architectural documentation up-to-date as their architecture naturally evolves. In this latter case, the architects might try to produce some informal, "quick-and-dirty" PowerPoint presentations describing their architectures just for the assessment. They might also try to present existing documentation that does not describe the actual architecture, but instead plans and procedures that document their original intended approach for producing the architecture.

12. **Take architecture maturity into account.**

    Lesson: It is important for the assessment team to know the status/maturity of the architectural elements being assessed.

    Rationale: For example, if an element's requirements are still largely unspecified, it is inappropriate to try to assess the quality of its architecture.

13. **Emphasize assessment over advice.**

    Lesson: Assess the architecture by listening to the architects' presentation of existing evidence and asking questions for clarification rather than spending significant amounts of time trying to train the architects in what is important for them to know.

    Rationale: The primary duty of the members of the assessment team is to assess the quality of the architecture. As technical experts, they will be tempted to try to solve any perceived deficiencies rather than just note them. There is insufficient time during as-

sessment meetings to solve architectural weaknesses, members of the assessment team probably do not have adequate depth of knowledge and experience in the subsystem being assessed to make optimal recommendations, and it is not their responsibility to architect the subsystem being assessed; that is the responsibility of the architecture team.

14. Ensure reasonable assessment size.

Lesson: The assessment should be decomposed into multiple, small assessments so that individual assessments can be performed in a reasonably small number of days.

Rationale:

a. The members of the architecture and assessment teams are typically extremely busy with other critical tasks.
b. They typically cannot afford to spend more than a few consecutive days supporting an assessment.
c. The assessments are typically locally located with the architects, and if the assessors must travel to the location of the assessments, assessments that last longer than a few days may require the assessors to stay over the weekends, thereby increasing assessment cost and imposing on the assessors.

7.4 Miscellaneous Lessons

1. Produce meeting preparation information.

To make the meetings more effective, it is important to ensure that the architecture team supplies preparatory materials to the assessment team and that the assessment team reviews this information. In one case, the assessment was made much more difficult because much of the documentation required as evidence was not supplied as early as originally required by the assessment method, as early as verbally promised by the contractor, or at all.

2. Decompose into two-person assessment subteams.

During the assessment meeting and subsequent development of the assessment report, the assessment team should be decomposed into two-person subteams, with each concentrating on different groups of quality factors. They should be assigned quality factors based on their experience, training, and expertise in the quality factors.

Rationale:

a. This use of subteams increases the productivity of the assessment team by enabling them to work in parallel, especially during the generation of the assessment report.
b. This also helps improve the completeness of the assessment by allowing evaluators to concentrate on evaluating the architecture's support for specific quality factors.

3. Provide appropriate preparatory information.

It is important to ensure that the information provided by the architecture team to the assessment team is appropriate. Appropriate information includes (1) general information needed to get up to speed on the architectural element being assessed, (2) architecturally significant requirements (i.e., requirements such as quality factor related requirements

that drive the architecture, and (3) official project documentation (e.g., architecture diagrams, models, and documents) that captures the architectural decisions that support the achievement of these architecturally significant requirements. Provide indexes and/or highlighting of relevant information buried within large amounts of preparatory evidentiary evidence. Preparatory information should not include planning or process documentation.

Rationale:

a. Plans for developing the architecture may show intent, but they do not demonstrate actual support for the architecturally significant requirements.
b. Procedures used to develop the architecture also do not demonstrate actual support for the architecturally significant requirements. Similarly, test plans and procedures do not clearly provide evidence for the architecture's support for testability.
c. Design and implementation documentation is typically not appropriate preparatory information because it does not directly address architectural decisions and their rationales. Any indirect architectural information that exists will be buried within large amounts of irrelevant information and be difficult to identify and interpret.
d. An index to relevant information or highlighting of relevant information makes it easier to identify the relevant information within the preparatory materials, especially evidentiary documentation.
e. Using indexes and highlighting enables assessors to avoid wasting time searching for relevant information or reading irrelevant information.

4. Take incremental architecture development into account.

In an iterative, incremental development process, it can be difficult to determine what architecture decisions apply to which blocks and releases.

5. Ensure adequate pre-meeting preparation.

The assessors cannot properly prepare for an assessment by reading the pre-assessment documentation unless it is made available with enough time to review before the assessment meeting.

6. Apply different emphasis at different levels.

It is difficult to obtain architectural information that supports a quality attribute assessment when starting at the top level of a large and complex software-intensive program or system. We need more than just an allocation of use case steps to very large domains and subsystems. We also need information about such topics as performance, safety, security, reliability, robustness, internationalization, etc. At the top levels, we need to see quality policies (e.g., safety policies at tiers 2–4) if the associated architectural mechanisms (e.g., safety mechanisms such as standard safeguards at tier 5) are at lower levels.

7. Differentiate observations and findings.

The assessment report should properly differentiate between observations and findings. The sections and subsections should be numbered so that it is easier to identify the topics of discussion.

8. **Differentiate architecture from design.**

   It is important to clarify the difference between architecture and detailed design and remain at the architecture level.

9. **Use scenarios as tests.**

   Use scenarios as test cases to verify that the architecture does in fact support specification compliance. Do **not** use scenarios to introduce and explain the basic architecture but rather to test the architecture's ability to fulfill its requirements and drivers after the architecture has been presented. *Review* the structural and functional architecture but *verify* the architecture via scenario test cases. Scenarios need to test the hardware/deployment architecture as well as the software architecture, especially with regard to fault tolerance, security data flow, performance, etc.

10. **Understand that not all tiers are equal.**

    Different architectural elements (e.g., subsystems) at the same tier in the architecture are typically not all of equal size, complexity, and criticality. For example, a small software configuration item and a major hardware/software subsystem might reside at the same level within the overall aggregation hierarchy of the system architecture. This means that the scope of the architecture assessments should not be restricted to all architectural elements at the same tier level (e.g., tier 4 or tier 5).

8 Future Directions

As with most endeavors, the QUASAR method is not perfect. There are several areas that deserve further research, lessons learned from trial use, and potential modifications to improve the method. These include

1. **Ensure architectural integrity across multiple subsystems.**

   The current QUASAR method emphases the assessment of system architectures in terms of the assessment of individual subsystem architectures, the assessment of these subsystem architectures in terms of the assessment of their sub-subsystem architectures, and so forth. When assessing the architectures of individual subsystems at the same tier in the overall system aggregation hierarchy, it becomes easy to miss the "whole forest for the individual trees."

   Specifically, it is important that higher level architects ensure the architectural integrity across multiple subsystems architected by different subsystem architecture teams working in parallel. For example, the different subsystem architects may have taken different architectural approaches to handling the scheduling of processes (e.g., the performance quality factor) or fault tolerance (e.g., the robustness quality factor). These approaches may work well at the individual subsystem level, but may cause significant problems that only show up when the subsystems are integrated (e.g., during integration or system testing).

   The assessment teams must take care to properly assess whether architectural integrity is maintained across the entire system architecture. If not, then the assessment team must determine if there are good reasons why the architectural approaches vary and that these variances do not cause unexpected or unacceptable architectural risks.

   It is possible that future versions of the QUASAR method will provide additional support for assessing architectural integrity across multiple subsystems within the overall system architecture.

2. **Weigh engineering tradeoffs between quality factors.**

   The current version of the QUASAR method emphasizes the assessment of the system architecture's support for cohesive sets of quality goals and requirements on an individual quality factor by quality factor basis. This is because quality cases consist of claims, arguments, and evidence associated with an individual quality factor or quality subfactor.

   Yet, a system architect must make engineering tradeoffs between the different quality factors. For example, increasing security typically makes it harder to achieve perform-

ance and usability requirements. Thus, requirements engineers must specify a feasible combination of quality requirements and architects use their limited resources to develop a globally optimal (read sufficient) architecture that adequately supports the meeting of all quality requirements rather than develop an over-architected architecture that is locally optimal for some quality factors and inadequate for others. The current version of QUASAR addresses this problem during the subsystem architecture assessment preparation and meeting in two ways:

a. as part of the architects' introduction of their architecture to the assessment team
b. as a rationale associated with one or more individual architectural decisions in quality case arguments

It is possible that future versions of the QUASAR method will provide additional support or more emphasis for assessing the architects' engineering tradeoffs between quality factors.

3. Increase emphasis on active probing of the architecture.

The current version of QUASAR emphasizes the architects' responsibility to make their cases that their architectures have sufficient quality. After all, they should best know what quality requirements their architecture must support (claims), what architectural decisions they made and why (arguments), and where they documented these decisions (evidence). However, architects naturally tend to emphasize the best aspects of their architecture. Architects do not tend to bring out the weaknesses in their architectures and naturally do not address potential architectural decisions and risks that did not occur to them. The current version of the QUASAR method recognizes this problem by assigning the assessment team the responsibility to ask the architects probing questions about their architectures. Currently, the assessment team can request the architects to run through one or more scenarios as a test of their architecture's support for a quality requirement.

It is possible that future versions of the QUASAR method will provide additional emphasis on the assessment team's responsibility to actively probe the architecture.

4. Develop a catalog of architectural styles, patterns, and mechanisms to use as standardized arguments.

In practice, there are relatively standard ways to solve commonly occurring architectural problems. Given a set of quality requirements, architects learn to consider associated architectural styles, patterns, and mechanisms to support their achievement. For example, hardware redundancy is a common way of improving availability and reliability. Therefore, it would be useful to develop a catalog of standard architectural arguments (i.e., architectural decisions and associated rationales) and associated evidence. This would help architects both develop higher quality architectures and build better quality cases. It would also help members of assessment teams understand what to expect from the architects' quality cases and ask probing questions when standardized solutions are not used.

5. Develop processes for achieving and determining sufficient architectural quality.

One of the most difficult problems for architects and assessors alike is to know when an architecture possesses sufficient quality. Because the quality of the architecture by itself does not determine the quality of the system, a quality architecture is a necessary but not sufficient condition for ensuring the quality of the system. A low-quality design or implementation can easily undo an architect's good work when it comes to ensuring the quality of a system.

The problem of knowing when an architecture is "good enough" is exacerbated by two contributing problems:

- In current practice, quality requirements are often poorly specified as vague goals. If the quality requirement does not have an associated minimum acceptable threshold on some scale of measure, then it is ambiguous and it does not specify (for the architect, assessor, and tester) how good is good enough.
- The architect cannot address individual quality requirements and quality factors in isolation. Systems must meet multiple types of quality requirements (e.g., performance and reliability, security and usability) that are naturally incompatible in the sense that making architectural decisions that increase one quality factor naturally decrease the other. For example, improving the availability, reliability, and robustness of a system typically decreases that system's performance. Thus, architects should not locally optimize a system's quality, one quality factor at a time; rather, architects must make engineering tradeoffs that achieve a globally optimized (actually acceptable) architecture.

Thus, it is difficult for both architects and assessors to know when a system's architecture sufficiently supports the system's ability to meet all of its quality requirements of all types (i.e., of all relevant quality factors). Currently, the QUASAR method relies on the combined experience of the members of the assessment team to come to a consensus on the sufficiency of the architecture. More work needs to be done to enable the QUASAR method to better support determining the sufficiency of the quality of the system/subsystem architectures.

6. Augment with books and training.

An earlier version of the QUASAR method was documented in a short, project-specific procedure document. This much larger handbook provides significant details and examples that could not be provided in the original procedure document. Yet this handbook is like a reference book and is not meant to be read from beginning to end as a normal systems engineering technical report or textbook. Therefore, this handbook should probably be augmented with a typical systems engineering technical book, which can be used as a textbook when teaching the quality assessment of system architectures. To improve stakeholder understanding of the QUASAR method, this handbook should also be supplemented with actual training courses covering both theory (e.g., quality cases and components of the QUASAR method) and practice (e.g., exercises in producing and presenting quality cases).

7. Consider tailoring down for smaller, simpler systems?

The QUASAR method was originally developed for and used on an extremely large and complex, software-intensive systems of systems. Thus, the vast majority of systems are significantly smaller and less complex. The question naturally arises as to if and how the QUASAR method should be tailored for use on such systems.

The QUASAR method is modular in the sense that it assesses the quality of the architecture of systems in terms of the quality of the architectures of the system's subsystems, the quality of the architectures of the subsystems in terms of the quality of the architectures of their sub-subsystems, and so on. Therefore, in theory, QUASAR should be equally applicable for use on relatively small and simple systems; one merely decreases the scope of the assessment by performing the subsystem assessments on fewer subsystems and against support for fewer quality attributes. In fact, even though an early version of the QUASAR method was applied to a very large and complex system of systems, the architectures assessed were of subsystems that were of moderate size and complexity.

A similar argument can be based on the large size and complexity of this handbook. However, this argument is probably misleading. One can provide a procedure document giving a brief overview of the QUASAR method in as little as 10 to 20 pages. The reason why this handbook is so large is because it provides an in-depth description of all aspects of the method as well as an extensive number of example quality cases.

Therefore, downsizing the QUASAR method is probably not a significant issue but is included here for the sake of completeness. More experience with using the method will determine whether or not this is a significant issue.

8. Expand beyond system architectures.

The current version of the QUASAR method is for assessing the quality of system architectures. But in order to do this, subsystem requirements reviews have been added to ensure that adequate derived quality requirements have been engineered (e.g., identified, analyzed, specified, and managed) and allocated to the subsystems, the architecture of which is being assessed. Thus, although the QUASAR method does not include the engineering of quality requirements, it does include a review to ensure that they are engineered in time to drive the architecture.

However, the quality of a system's architecture does not guarantee the quality of the associated system. For example, a system may have proper reliability requirements and have a good architecture that incorporates architectural patterns and mechanisms that support the reliability of the system. But if the system's design, implementation, and testing include a sufficient number of specific kinds of defects, then the system will *not* meet its reliability requirements. A quality architecture is necessary but not sufficient to produce a quality system.

Therefore, what is eventually needed is a system engineering approach to engineer quality into the system and an assessment approach to assess the quality of the system's re-

quirements, architecture, design, implementation (and production), and installation (including for instance, its configuration). In addition to its architecture, quality cases can be used to document the quality of a system's requirements, design, implementation, and installation. Therefore, the QUASAR method could be expanded beyond system architecture quality assessments to the quality assessment of the system itself and all of its intermediate work products.

9 Conclusion

You should take away the following observations and ideas from the information in this handbook:

- The quality of a system's architecture is a critical driver of the quality of the system and is thereby critical to the success of the system.
- It is imperative to assess the quality of a system's architecture. Specifically, it is important to incrementally assess the quality of the system's architecture as the architecture is incrementally developed. In this way, architectural defects can be fixed and architectural risks can be managed before so much design and implementation takes place that correcting the architecture becomes impractical.
- System architects know (or should know) the following about their architectures:
  - the architecturally significant quality goals and requirements that were derived and allocated to their system or subsystems and that therefore drove their architectural decisions
  - whether their architectures adequately support the system achieving its architecturally significant quality goals and requirements
  - the architectural decisions they have made and why they made them
  - how and where they documented these architectural decisions
- Quality cases are a good way for the system architects to organize and present information and thereby are a good way to make their case to the assessors that their architectures have sufficient quality.
  - Architectural quality cases consist of a cohesive set of
    - claims that the architecture sufficiently supports the system or subsystem's derived and allocated quality goals and requirements
    - clear and compelling arguments (consisting of architecture decisions and associated rationales) justifying belief in the architects' claims
    - sufficient evidence (consisting of official diagrams, models, and documents or assessor-witnessed demonstrations) supporting the architect's arguments
- This handbook provides numerous examples to give architects guidance as to what their architectural quality cases should look like.
- QUASAR is a system architecture quality assessment method that is used to assess the quality of a system's architecture based on quality cases developed and presented by the system architects to the assessment team.

- The QUASAR method includes
  - teams and member roles with associated responsibilities
  - four phases consisting of associated tasks and component steps
  - work products that are produced and used by members of these teams during the QUASAR phases and tasks
- The QUASAR method was largely (but not totally) based on experience gained and lessons learned during its use in the assessment of the architecture of an actual large and complex software-intensive system.

Appendix A Acronyms and Abbreviations

AC	Alternating Current
APM	Automated People Mover
ASCII	American Standard Code for Information Interchange
ASP	Acquisition Support Program
ATS	Automated Taxi Subsystem
CDR	Critical Design Review
C	Component
CI	Configuration Item
COMSEC	Communications Security
COMPUSEC	Computer Security
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
DNS	Domain Name Service
DoD	Department of Defense
DoS	Denial of Service
DS	Door Subsystem
EMSEC	Emissions Security
FDDI	Fiber Distributed Data Interface
FDCS	Fire Detection and Control Subsystem
FDSS	Fire Detection and Suppression Sub-Subsystem
FTP	File Transfer Protocol

GSN	Goal Structuring Notation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
ICMP	Internet Control Message Protocol
INFOSEC	Information Security
I/O	Input/Output
IP	Internet Protocol
IPT	Integrated Product Team
ISDN	Integrated Services Digital Network
JPO	Joint Program Office
JSF	Joint Strike Fighter
MHz	Megahertz
MPEG	Moving Picture Experts Group
MTBF	Mean Time Between Failures
MTBMCF	Mean Time Between Mission-Critical Failures
NETSEC	Network Security
NFS	Network File System
OPSEC	Operations Security
OS	Operating System
OTS	Off-the-Shelf
PBS	Power Braking Sub-Subsystem
PDR	Preliminary Design Review
PERSEC	Personal Security
PHYSEC	Physical Security

PMO	Program Management Office
PPP	Point-to-Point Protocol
QUASAR	Quality Assessment of Software Architectures
RFA	Request for Action
RFI	Request for Information
RFP	Request for Proposal
RMS	Rate-Monotonic Scheduling
SEI	Software Engineering Institute
SME	Subject Matter Expert
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SS	Sensor Sub-Subsystem
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol
UML	Unified Modeling Language
VDC	Volts of Direct Current
VHF	Very High Frequency
WSDL	Web Services Description Language
XML	Extensible Markup Language

Appendix B Glossary

Allocated Requirement

a requirement that is [partially] assigned to a subsystem

Architect

the role that filled by a person who develops the architecture of a system or one of its subsystems

Architectural Decision

a significant decision made by one or more architects that determines a part of the architecture, such as the selection of an architectural style, pattern, or mechanism

Architecturally Significant Requirement

a system or subsystem functional, data, interface, or quality[51] requirement that significantly influences the architecture of the associated system or subsystem

Not every requirement has a meaningful impact on the architecture, because the architecture would be essentially the same regardless of the existence of the requirement. On the other hand, other requirements drive the architects to make significant architectural decisions. In fact, such architecturally significant requirements are the primary reasons why architects choose to use a specific architectural style, pattern, or mechanism. In other words, architecturally significant requirements are those requirements that cause the architects to incorporate important, pervasive, top-level, and global decisions and inventions into their architectures. Because such architecturally significant requirements drive the architecture, they become the requirements against which the architecture is assessed. Quality cases demonstrate that the system/subsystem architecture sufficiently supports the ability of the system/subsystem to meet its architecturally significant requirements, and QUASAR assessments use quality cases to determine whether the system/subsystem architecture sufficiently supports the ability of the system/subsystem to meet its architecturally significant requirements.

Architecture

the most important, pervasive, top-level, and therefore global (i.e., strategic) engineering decisions, inventions, styles, patterns, and mechanisms and their associated rationales

Architecture primarily includes the overall system structure in terms of its essential elements, their relationships, and their associated characteristics and behavior that enable them to collaborate together to achieve their derived and allocated requirements. Architecture also deals with all decisions that have a global impact such as the

[51] Quality requirements tend to have the biggest influence on the architecture, so many if not most of the architecturally significant requirements are quality requirements.

- technology, languages, and off-the-shelf products to be used
- *standard* approaches to concurrency (e.g., scheduling approach and ways of avoiding concurrency defects such as race conditions, starvation, and deadlock) to be used
- *standard* approaches to achieve robustness and deal with exception raising, propagation, and handling

The architecture is the most important work product produced during the performance of architecture tasks and provides a blueprint for design, implementation, and testing.

This definition is quite general, while still clearly differentiating architecture from design. Architecture is global, whereas design is local. Architecture deals with strategic issues, whereas design deals with tactical issues. Architecture involves ensuring the integrity of the big-picture vision, whereas design deals with the lower level details. Finally, architecture involves choices that affect many developers, whereas design decisions impact few beyond the individual designer and implementer of that design.

Architecture Argument

a logically coherent series of clear and compelling valid reasons justifying the assessors' beliefs in one or more architecture claims. An architecture argument is typically on of the architects' architectural decisions (e.g., the use of appropriate architectural components, mechanisms, or patterns).

For example, the use of redundancy can be used as an argument that a system fulfills its availability and reliability requirements, the use of open interface standards for key interfaces can be used as an argument of interoperability, and the use of a firewall can be used as an argument that a system fulfills its security requirements.

Architecture Assessment

a determination of the quality of the architecture of a system or subsystem

Architecture Claim

an assertion made by an architect as part of an architecture quality case that the architecture of a system/subsystem supports the achievement of one or more of its architecturally significant goals or the meeting of one or more architecturally significant requirements

Architecture Evidence

official, factual information that is part of a architecture quality case that clearly proves the truth of the architects' arguments supporting their architectural claims

Architecture Quality Case

a quality case that makes the case for the quality of an architecture and therefore consists of architectural claims, architectural arguments, and architectural evidence

Architecture Team

a team responsible for producing an architecture and making a case to the assessment team that their architecture adequately supports each type of architecturally significant requirements

Argument

a logically coherent series of clear, compelling, and valid reasons for believing one or more claims that a system or subsystem fulfills one or more of its architecturally significant requirements, which is given as part of a quality case

For example, the use of redundancy can be used as an argument that a system fulfills its availability and reliability requirements, the use of open interface standards for key interfaces can be used as an argument of interoperability, and the use of a firewall can be used as an argument that a system fulfills its security requirements.

Assessment

an assessment during which certain system characteristics or qualities are determined

Assessment Kickoff Meeting

the initial meeting during which the architecture team and assessment team come to a consensus on the scope and general scheduling of the following assessment meetings

Assessment Meeting

a meeting during which the architecture team presents their case to the assessment team that their architecture adequately supports each type of architecturally significant requirements and answers questions posed to them by members of the assessment team

Assessment Team

the team that is responsible for assessing the quality of the architecture based on the architecture team's claims, arguments, and evidence

Claim

an assertion made as part of a quality case that a system/subsystem achieves one or more goals or meets one or more requirements

Note that a claim must be supported by clear and compelling arguments that in turn are based on sufficient official evidence in order to be judged substantiated by the assessment team.

Derived Requirement

a requirement not explicitly specified by the customer but rather engineered and explicitly specified by the requirement team in order to fulfill one or more of the customer requirements

Evidence

official, factual information that is part of a quality case that clearly proves the truth of the architects' arguments supporting their claims that a system or subsystem fulfills one or more of its architecturally significant requirements

Although it typically consists of current project architectural diagrams, models, and textual documentation that are under configuration control, evidence may also include hardware exhibited to and directly observed by the assessment team (e.g., the configuration of the subsystems of a system prototype seen during a tour of a development lab).

For example, a current official architecture diagram that clearly shows the incorporation of multiple servers and paired networks is valid evidence supporting the architect's argument that the architecture incorporates the use of redundant hardware which in turn

backs up the architect's claim that the architecture provides adequate availability and reliability.

Note that architecture plans and development procedures should *not* be considered to be evidence because they do not prove actual architecture support for architecturally significant requirements, only intent.

Goal

1. a perceived need of a legitimate stakeholder in the system that drives the identification, analysis, and specification of the requirements
   In other words, goals are the ambiguous, infeasible, unverifiable, informally documented, architecturally significant desires that are all too often specified instead of the specific requirements that should have actually been engineered. For example, "The system shall be modifiable, reliable, safe, and secure."
2. a top-level purpose that is decomposed into one or more detailed objectives
   For example, the *goals* listed in the first section of this handbook that drive the objectives of the QUASAR method.

Method

a cohesive collection of endeavor-specific method components (e.g., work products, work units such as tasks, and the teams and roles that perform them) that model a systematic *intended* way of producing work products and providing services
For example, QUASAR is a method for performing system architecture assessments.

Model

an abstraction (simplification) of something that captures its essential characteristics for some specific purpose while ignoring unimportant or diversionary details

Quality

the degree to which a work product (e.g., system, subsystem, architecture, or document) has useful and desirable characteristics as represented by its associated quality factors

Quality Case

for a given quality factor,[52] the combination of a

- cohesive collection of one or more *claims* that the system or subsystem architecture[53] adequately fulfills its associated quality-related requirements
- corresponding structured set of clear and compelling valid *arguments* supporting these claims
- sufficient amount of properly documented *evidence* supporting these arguments to convince a skeptical assessor of the validity of the claims and arguments

---

[52] A quality case is a generalization of a safety case to other quality factors. There are various kinds of quality cases (e.g., interoperability cases, maintainability cases, reliability cases, and safety cases).

[53] Because this handbook deals with system architecture assessments, we only address system or subsystem architecture-level quality cases. In general, quality cases could include design and implementation arguments and evidence.

Quality Case Diagram

a UML class diagram that summarizes the claims, arguments, and evidence comprising a quality case and the relationships between them

Quality Criterion

a specific description of something that provides evidence either for or against the existence of a specific quality factor or subfactor

Quality Factor (quality attribute, quality characteristic, or "-ility")

an important attribute, characteristic, or property of a work product (e.g., system, subsystem, architecture) or process that characterizes a part of its overall quality

Quality Measure

a unit of measure that provides a range of numerical values enabling the measurement of the quality of a work product or process by documenting the degree to which the work product or process possesses (or shall possess) a specific quality factor or quality subfactor

Quality Requirement

a requirement specifying that a system or subsystem must have a minimum required amount of a quality factor or one or more quality factors

A quality requirement specifies that under certain conditions, the system or subsystem must exhibit a quality criterion demonstrating that one or more associated quality subfactors exist beyond a minimum threshold on an associated quality measure.

The following is an example of a quality (performance) requirement: "When not in degraded mode (*condition*), the mortgage processing system shall correctly process mortgage applications (*quality criterion*) with a throughput (*performance quality subfactor*) of at least 100 applications per second (*threshold on quality measure*)."

Quality-Related Requirement

a requirement that has ramifications for the achievement of the associated quality

Four types of quality-related requirements include quality requirements (safety requirements), quality-significant requirements (e.g., functional, data, or interface requirements that make the system safety critical because they can lead to hazards and accidents if not implemented correctly), quality subsystem requirements (e.g., requirements for safety subsystems such as fire detection and suppression systems), and quality constraints (e.g., architecture, design, and implementation decisions that are to be treated as if they were requirements).

Quality Subfactor

an important part of a quality factor

For example, jitter, latency, response time, schedulability, and throughput are quality subfactors of the quality factor performance, whereas accidental harm, accident, hazard, and safety risk are quality subfactors of safety.

QUASAR (Quality Assessment of System ARchitectures)

a method for assessing the quality of a system's architecture in terms of the architecture's support for its associated architecturally significant requirements based on the architects' claims, supporting arguments, and underlying evidence

Requirement

an established need justifying the timely allocation of resources to develop a capability to achieve approved goals or accomplish approved missions or tasks

Unlike goals, requirements should be cohesive, complete, consistent, correct, current, externally observable, feasible, mandatory, relevant, stakeholder-oriented, unambiguous, validatable, and verifiable.

Requirements Meeting

a subsystem-level meeting during which the architecture team demonstrates that the architecturally significant requirements have been adequately engineered and identified to enable them to engineer the architecture

Requirements Team

the team that engineers the requirements (including architecturally significant requirements) for a system or one or more of its subsystems

Safety Case

a cohesive collection of claims that a system is sufficiently safe for a given usage in a given environment (i.e., that it fulfills its safety-related requirements), whereby the claims are based on a corresponding structured set of clear and compelling valid arguments that are supported by a sufficient amount of properly documented evidence

Software-Intensive System

a system, major functionality and characteristics of which are implemented via software as opposed to via hardware or manual procedures

Subsystem

an integrated subset of a system that provides a capability that is essential to the success of the system

System

a major, functionally cohesive, executable, and integrated aggregation of components (including hardware, software, and potentially data, manual procedures, and facilities) that collaborate to provide the capability to perform one or related missions

System Architecture

an architecture of a system capturing its most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., subsystems, sub-subsystems, and their relationships) and associated characteristics and behavior including how they collaborate together to achieve their allocated requirements

Appendix C Quality

The QUASAR method is fundamentally founded on the concept of quality. It assess the *quality* of system architectures in terms of *quality* cases, and the first part of quality cases are claims that the architecture sufficiently supports the system's achievement of *quality* goals and meeting of *quality* requirements. Clearly, the QUASAR method cannot be effectively used unless there is a firm and unambiguous understanding of the meaning of quality.

C.1 Quality Model

The term *quality* is quite complex and often means different things to different people. To avoid this ambiguity, systems engineers must use a quality model that makes the term "quality" specific and useful for engineering systems [Firesmith 03]. A quality model does this by decomposing the term into its component concepts and their relationships to one another.

As illustrated in Figure 20, a *quality model* is a hierarchical model (i.e., a collection of related abstractions or simplifications) for formalizing the concept of the quality of a system in terms of its following components:

- **Quality Factor (aka, quality attribute, quality characteristic, "-ility")**

  A *quality factor* is a high-level characteristic, attribute, or property of a system or subsystem that characterizes an aspect of its quality. Quality has to do with the degree to which the system or subsystem possesses a combination of characteristics, attributes, aspects, or traits that are desirable to its stakeholders. As listed in Section C.2, there are many different quality factors such as availability, extensibility, performance, reliability, reusability, safety, security, and usability. These factors determine whether or not as system has sufficiently high quality. Because many of the quality factors end in the letters "-ility," they are often collectively referred to as the "-ilities." Quality factors can be classified into more specific subclasses of quality factors (e.g., safety is a kind of defensibility, which is a kind of dependability, which is a kind of quality).

- **Quality Subfactor**

  A *quality subfactor* is a major component (aggregation) of a quality factor or of another quality subfactor. Thus, throughput and response time are quality subfactors of performance, whereas internationalization and personalization are quality subfactors of configurability.

- **Quality Measure**

  Quality measures are measurement scales that provide a way to measure and quantify a quality criterion. Quality measures thus make quality criteria objective and unambiguous. Quality measures support the production of metrics by providing numerical values for specifying or estimating the quality of a system or subsystem by measuring the degree to which it possesses a specific quality factor or subfactor.

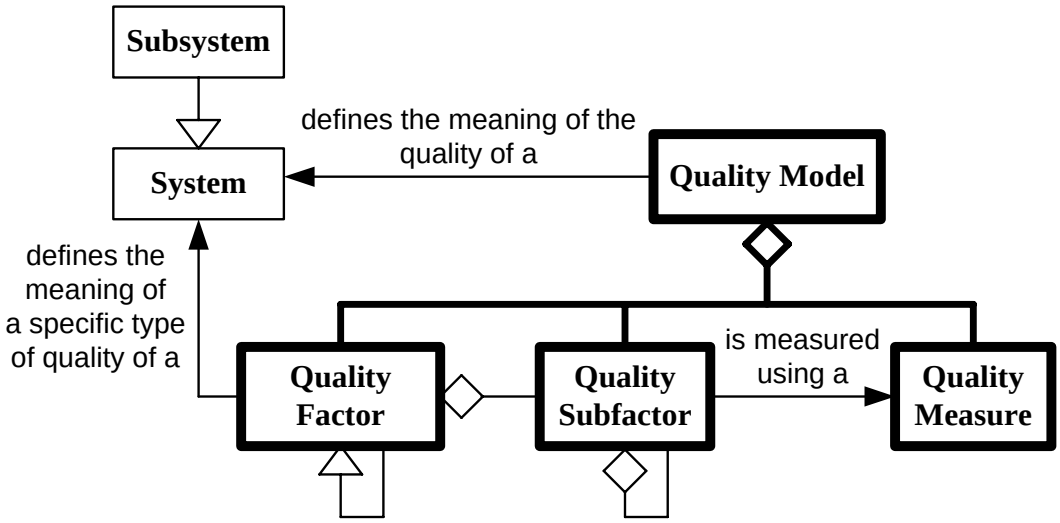

Figure 20: Quality Model

C.2 Example Quality Factors and Subfactors

Quality factors and subfactors are used to produce quality requirements, which are a major type of architecturally significant requirement. QUASAR assessments are thus typically organized around assessing architectural support for these quality factors and subfactors.

Unfortunately, a great deal of time can be wasted in fruitless arguments between and among members of the architecture, assessment, and requirements teams regarding the meaning of the quality factors used to produce quality requirements, drive the system and subsystem architectures, and on which the system quality assessments are based. There are several standard quality models [ISO 91] and associated taxonomies and ontologies [Firesmith 03]. As illustrated in Figure 21, the following examples taken primarily from the OPEN Process Framework Repository provide a fairly complete and consistent hierarchy of quality factors [OPFRO 06].

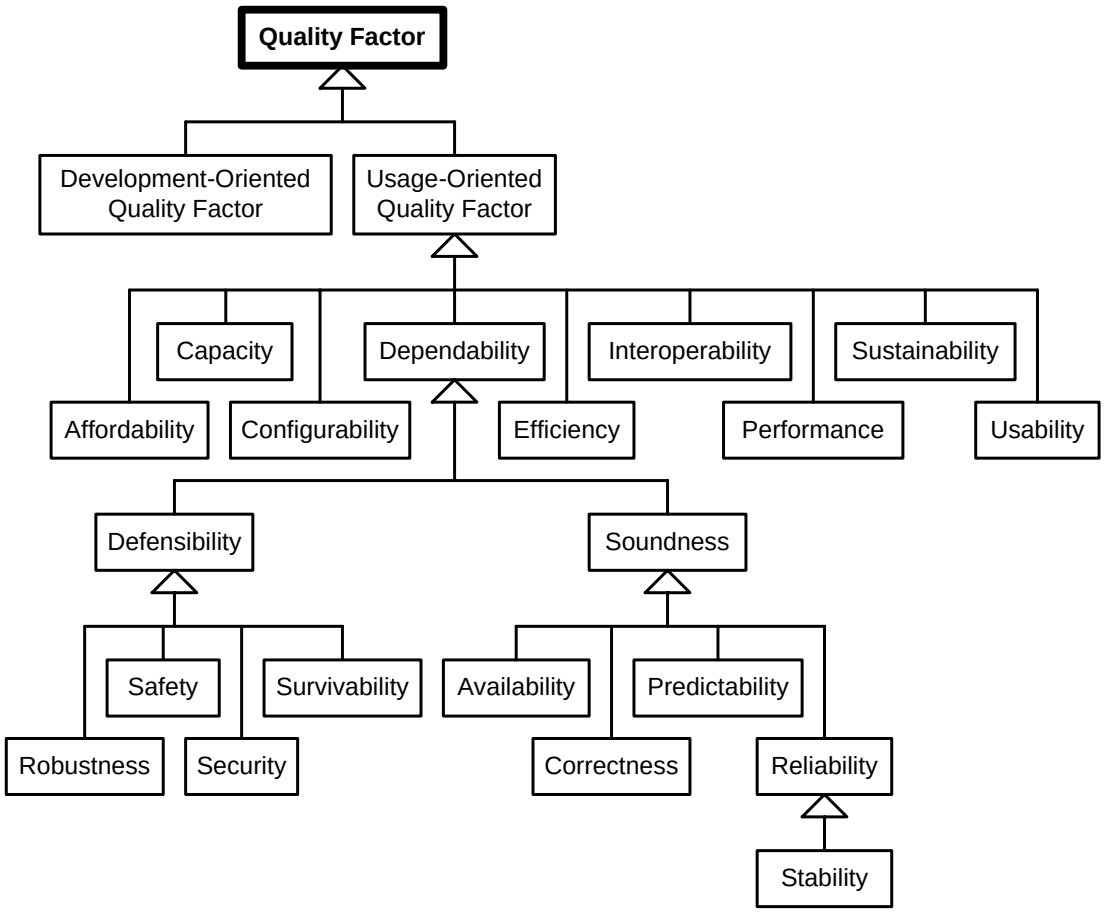

Figure 21: Hierarchy of Usage-Oriented Quality Factors

- **Affordability** – the degree to which a system or subsystem can be developed and manufactured within budget

  Quality subfactors of affordability include
  - *Development Cost* – the cost to develop the system or subsystem
  - *Manufacturing Cost* – the cost to manufacture instances of the system
  - *Support Cost* (aka, sustainment cost) – the cost to support the system one manufactured and delivered.
  - *Retirement Cost* (aka, disposal cost) – the cost to retire the system once it is no longer needed.
- **Availability** (aka, readiness) – the degree to which a system or subsystem is *ready* to function without failure in one or more specified ways at any time during a specified period of time under normal conditions or circumstances (i.e., the proportion of time that the system or subsystem can be used)
- **Capacity** – the degree to which a system or subsystem can successfully handle a large number of things at a single point in time or during a specific interval of time
- **Compliance** – the degree to which a system or subsystem adheres to related standards, conventions, regulations in laws, and similar prescriptions

- **Configurability** – the degree to which a system or subsystem is or can be configured into multiple forms (i.e., configurations)

  Quality subfactors of configurability include

  - *Internationalization* (aka, globalization and localization) – the degree to which the system can be (or is) configured to function appropriately in a global environment in terms of
    - native languages, language idioms, spelling, and character sets
    - formats of contact information such as name, address, and phone number
    - currencies including real-time currency conversion
    - legal issues such as import/export laws, tariff and sales tax calculations, customs documentation, trademarks, and privacy laws
    - culture (e.g., use of inappropriate colors, symbols, or product names)
  - *Personalization* – the degree to which the system can be (or is) configured so that individual users can be presented with a unique user-specific experience
  - *Subsetability* – the degree to which the system can be released in multiple variants, each of which implements a different subset of requirements (i.e., each variant is a subset of a primary complete variant)
  - *Variability* – the extent to which the system exists in multiple variants, each of which implements a different superset of the common set of requirements (i.e., some requirements are unique to each variant)
- **Consistency** – the degree to which the components of a system or subsystem:[54]
  - belong to the same architectural styles
  - implement the same architectural patterns
  - use the same architectural mechanisms
- **Correctness** – the degree to which a system or subsystem and its outputs are free from defects

  Quality subfactors of correctness include

  - *Accuracy* – the magnitude of defects (i.e., the deviation of the actual or average measurements from their true value) in the system's stored and output quantitative data
  - *Latent Defects* – the degree to which the system is free from defects upon delivery
  - *Precision* – the degree of dispersion of the system's stored and output quantitative data around their average values
  - *Timeliness* – the degree to which data remains current (i.e., up-to-date)
- **Defensibility** – the degree to which a system or subsystem defends valuable assets from accidental or malicious harm (Defensibility can be classified into the quality factors: robustness, safety, security, and survivability.)

[54] For example, the components have the same structure and communicate the same way via similar interfaces.

- **Efficiency** – the degree to which a system or subsystem effectively uses (i.e., minimizes its consumption of) its resources (e.g., bandwidth, CPU [central processing unit] cycles, storage, electricity)
- **Interoperability** – the degree to which a system operates (i.e., collaborates and interfaces) effectively with specified [types of] external systems by successfully providing services and data to those systems and using services and data provided by those systems
- **Intraoperability** – the degree to which subsystems within a system operate effectively with other subsystems within the system by successfully providing services and data to those subsystems and using services and data provided by those subsystems
- **Maintainability** – the degree of ease[55] with which a system or subsystem can be modified between major releases when not required by changes to requirements. Maintainability can be classified into the quality factors: adaptive, corrective, perfective, and preventative maintainability.

  Quality subfactors of maintainability include

  – *Analyzability* – the degree of ease with which defects, deficiencies, and causes of failures can be diagnosed and localized to the components to be modified
  – *Modifiability* (aka, changeability) – the degree of ease with which a system or subsystem can have specified types of changes made
  – *Extensibility* – the degree of ease with which a system or subsystem can be enhanced to meet specified future goals and significantly changing requirements
  – *Scalability* – the degree of ease with which a system or subsystem can be modified to increase its existing capacities
  – *Verifiability* – the degree of ease with which changes to a system or subsystem can be verified as having been correctly made and to be without unexpected and undesirable side-effects
- **Operational Environment Compatibility** – the degree to which a system or subsystem can be used and functions correctly under specified conditions of the physical environment in which it is intended to operate
- **Performance** – the degree to which a system or subsystem operates within its designated temporal constraints

  Quality subfactors of performance include

  – *Jitter* – the degree to which the variability of the time intervals between system controlled periodic actions remains within its designated constraints

[55] The phrase “the degree of ease” refers to the amount of effort required to do something, whereas the phrase “the degree to which” refers to the extent to which something occurs. The difference between these two phrases determines the type of scale of measure used to set the threshold on the associated quality requirements.

- *Latency* – the degree to which the time that the system or subsystem takes to execute specific tasks (e.g., system operations and use case paths) from end to end is within acceptable time limits
- *Response Time* – the degree to which the time it takes for the system or subsystem to initially respond to a client request for a service is within acceptable time limits
- *Schedulability* – the degree to which events and behaviors are deterministic and can be accurately scheduled
- *Throughput* – the degree to which the system is able to complete an operation and provide a service within acceptable time limits

- **Portability** – the degree of ease with which a software system or subsystem can be moved to specified [types of] hardware (e.g., server) or software (e.g., operating system or middleware) environments

- **Predictability** – the degree to which the behavior of a system or subsystem is deterministic (i.e., predictable) for a given set of inputs when in a given state and/or environment

- **Producibility** – the degree of ease with which a system or subsystem can be produced (e.g., manufactured) to meet its requirements

- **Reliability** (aka, continuity) – the degree to which a system or subsystem *continues* to function without failure in one or more specified ways during a specified period of time under normal conditions or circumstances

- **Reusability**
  - Architect ***with*** reuse – the degree to which the current system or subsystem architecture enables externally produced components to be incorporated with little or no modification
  - Architect ***for*** reuse – the degree to which components currently being produced are enabled to be incorporated with little or no modification into the architecture of other specified systems
- **Robustness** – the degree to which a system or subsystem tolerates potentially harm-causing events or conditions and recovers from them

  Quality subfactors of robustness include
  - **Tolerance** – the degree to which a system or subsystem tolerates potentially harm-causing events or conditions
    - *Environmental Tolerance* – the degree to which essential mission-critical services continue to be provided in spite of potentially harm-causing *environmental conditions* (e.g., salt spray causing corrosion or radiation randomly changing the value of a bit within memory)
    - *Error Tolerance* – the degree to which essential mission-critical services continue to be provided in spite of the presence of *erroneous input* (e.g., incorrect, stale, or out-of-order data). Note that erroneous input is typically due to human error, although it may also be due to sensor failure, timing delays, etc.

  - *Fault Tolerance* – the degree to which essential mission-critical services continue to be provided in spite of the presence or execution of defects, whereby a defect (aka, fault and bug) is an underlying flaw in a work product (i.e., a work product that is inconsistent with its requirements, policies, goals, or the reasonable expectations of its customers or users). Note that a defect may or may not cause a failure depending on whether or not the defect is executed and whether or not exception handling prevents the failure from occurring.
  - *Failure Tolerance* – the degree to which the system continues to provide essential mission-critical services in spite of the occurrence of failures, whereby a failure is the execution of a defect that causes an inconsistency between an executable work product's actual (i.e., observed) and expected (e.g., specified) behavior.
  - *Fail Safety* – the degree to which the system places itself into a safe operating mode in the event of specific failures
  - *Fail Security* – the degree to which the system places itself into a secure operating mode in the event of specific failures
  - *Fail Softness* – the degree to which the system continues to provide partial operational capabilities (possibly in a degraded mode) in the event of specific failures
- **Incident Tolerance** – the degree to what a system or subsystem continues to provide essential mission-critical services in spite of the occurrence of incidents
  - *Safety Incident Tolerance* – the degree to which a system or subsystem continues to provide essential mission-critical services in spite of the occurrence of *safety incidents* (e.g., accidents and near misses)
  - *Security Incident Tolerance* – the degree to which a system or subsystem continues to provide essential mission-critical services in spite of the occurrence of *security incidents* (e.g., security attacks and probes)
  - *Survivability Incident Tolerance* – the degree to which a system or subsystem continues to provide essential mission-critical services in spite of the occurrence of *survivability incidents* (e.g., military attacks)
- **Recoverability** – the degree to which a system or subsystem recovers from a failure
  - *Functionality Recoverability* – the degree to which a system or subsystem reestablishes its prior level of functionality after a failure
  - *Data Recoverability* – the degree to which a system or subsystem recovers data directly affected by a failure
  - *Recovery Effort* – the amount of effort and time needed to reestablish a system or subsystem's functionality and to recover any data that has been directly affected by a failure

- **Soundness** – the degree to which a system or subsystem is sound. Soundness can be classified into the quality factors: availability, correctness, predictability, and reliability.

- **Safety** – the degree to which
  - a system or subsystem prevents or reduces in probability or severity, detects, reacts to, and adapts to
    - accidental (i.e., unplanned and unintended but not necessarily unexpected) harm to valuable assets
    - safety incidents (i.e., accidents and near misses)
    - hazards (i.e., unsafe conditions)
  - safety risks associated with the system or subsystem are acceptably low

- **Security** – the degree to which
  - a system or subsystem prevents or reduces in probability or severity, detects, reacts to, and adapts to
    - malicious (i.e., unplanned and unintended but not necessarily unexpected) harm to valuable assets caused by attackers
    - security incidents (i.e., attacks and probes)
    - threats (i.e., existence of attackers with means, motives, and opportunities)
  - security risks associated with the system or subsystem are acceptably low
- **Stability** – the degree to which a system or subsystem continues to deliver *mission-critical* services during a given time period[56] under a given operational profile regardless of any failures whereby the
  - failures may prevent the system from delivering less critical services
  - failures limiting the delivery of mission-critical services occur at unpredictable times
  - root causes of such failures are difficult to identify efficiently
- **Sustainability** – the degree of ease with which a system or subsystem can be supported once placed into use (i.e., fielded into its operational environment)
- **Testability** – the degree of ease with which a system or subsystem facilitates the creation and execution of successful tests (i.e., tests that cause failures due to underlying defects[57])

  Quality subfactors of testability include
  - *Controllability* – the ease with which the system can be
    - placed into the proper pretest state
    - stimulated with the test message or data
  - *Observability* – the ease with which the system can be observed to
    - be in the proper pretest state
    - provide the proper output to its clients, peers, and servers (e.g., returned values, output messages, output requests for data)
    - in the proper posttest state
- **Usability** – the degree to which the system's human user interface enables a specified group of users achieve specified goals in a specified context of use

---

[56] Stability is a type of reliability. Whereas reliability is typically measure in terms of mean time between failures (MTBF), stability is typically measured in terms of mean time between mission-critical failure (MTBMCF). Stability is closely related to robustness, which is a type of defensibility. Whereas robustness refers to how well a system defends itself in terms of its tolerance of negative events and conditions and in terms of its recoverability once failure occurs, stability refers to how often mission-critical failures occur.

[57] See http://www.opfro.org/Glossary/GlossaryF.html for definition of "failure."

Quality subfactors of usability include

- *Attractiveness* (aka, engagability, preference, and stickiness[58]) – the degree to which users find the system to
  - be attractive or appealing
  - engage their attention
  - provide a positive user experience
  - be preferable to its alternatives
  - make them to continue to use it
  - make them return to use it in the future
- *Credibility* (aka, trustworthiness) – the degree to which users are confident with and have trust in the system including that its
  - output and behavior are correct
  - content is authoritative
  - owner's motives are trustworthy
  - developers are competent
- *Differentiation* – the degree to which the system differentiates itself from competing products
- *Ease to Entry* – the ease with which users can start using the system (e.g., can log on and begin using their desired functionality without waiting an excessive amount of time to be identified, authenticated, and navigate to the point where they can start performing their tasks)
- *Ease of Location* – the ease with which users can find the system's content or services (e.g., finding Web applications such as Web sites using search engines)
- *Ease of Remembering* – either the degree to which occasional users can remember how to use the system to perform common tasks or the degree to which regular users to can remember how to use the system to perform infrequent tasks
- *Effectiveness* (aka, operability) – the degree to which the system its enables users to successfully achieve their goals
- *Error Minimization* – the degree to which the system minimizes the number of errors that its users make
- *Learnability* – the degree to which representative users can learn to use the system to achieve their goals (e.g., to find desired content and to perform their tasks)
- *Navigability* – the degree to which the product enables users to easily move through the user interface or documentation to achieve their goals
- *Operability* – the degree to which the system minimizes the amount of effort users (and operators) must expend to achieve their goals (in relation to the accuracy and completeness with which these goals are achieved)
- *Retrievability* – the ease with which the product enables users to obtain information in a form that is useful to them (e.g., print out a paper report, make a copy of a multimedia file)

[58] The term "stickiness" is typically used with reference to Web pages and refers to how long users remain at (i.e., remain stuck to) given Web pages.

- *Suitability* (aka, appropriateness) – the degree to which users find that the product to be suitable for the performance of their tasks
- *Understandability* – the degree to which users find the system's human interfaces and output to be clear, legible, unambiguous, and comprehensible (especially during unusual situations)
- *User-Satisfaction* – the degree to which users are satisfied with the product and consider it to be beneficial to them

C.3 Quality Requirements

Having quality requirements that themselves exhibit high quality is a critical prerequisite for the production of a high-quality system architecture that sufficiently supports the systems' meeting of these quality requirements. Building on Figure 20, which illustrates the structure of a quality model in terms of its quality factors, quality subfactors, and quality measures, this subsection defines quality requirements and shows how they relate to the components of the quality model.

As illustrated in Figure 22, a quality requirement consists of a three main parts.

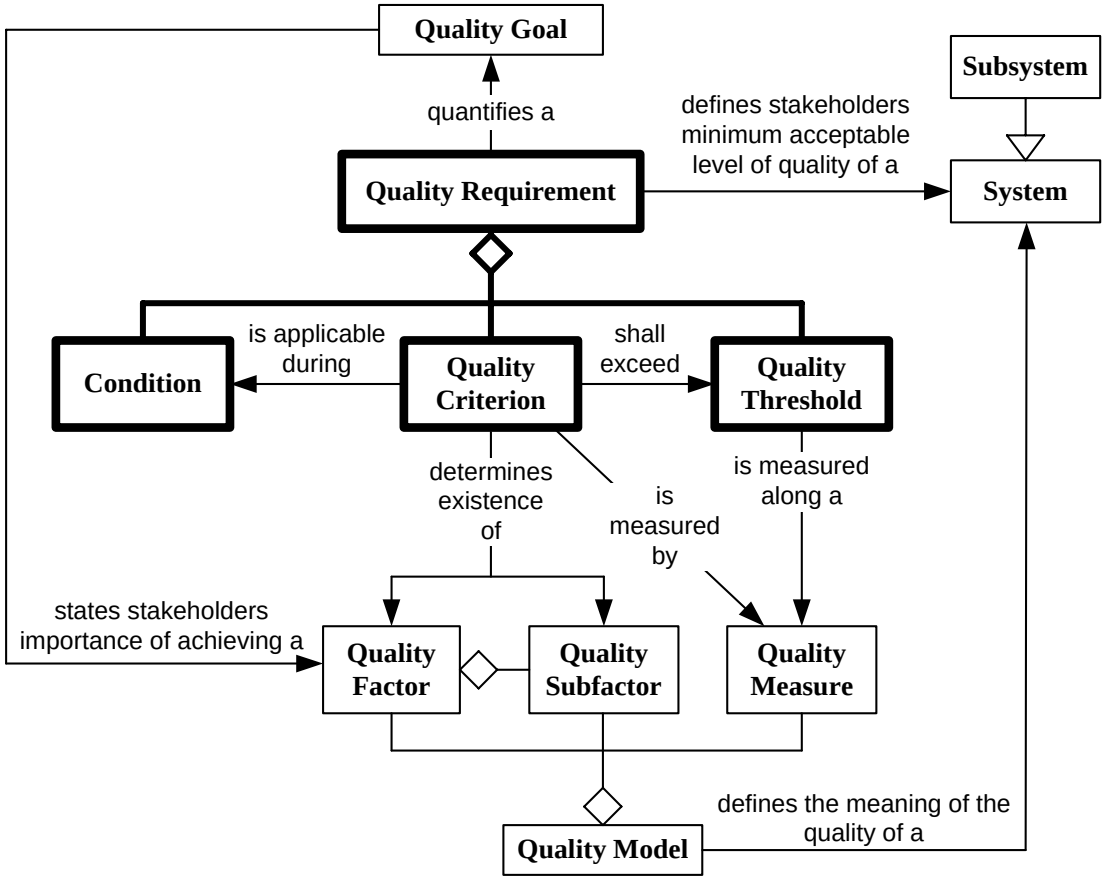

Figure 22: Components of a Quality Requirement

- **Condition**

  An optional *condition* that states under what conditions the quality requirement must be met. For example, quality requirements specifying high performance and reliability may only hold during normal conditions, but not under degraded mode operations.

- **Quality Criterion**

  A *quality criterion* is a system-specific description that provides evidence either for or against the existence of a given quality factor or subfactor.[59] Quality criteria significantly contribute toward making the high-level quality factors and subfactors detailed enough to be unambiguous and verifiable. When quality criteria are adequately specific, they lack only the addition of quality metrics to make them sufficiently complete and detailed to form the basis for detailed quality requirements. If quality is the trunk of the tree and the quality factors and subfactors are the branches and twigs, then quality criteria are the leaves. There are many more quality criteria than quality factors and subfactors because there are typically numerous criteria per factor and subfactor. Quality criteria are also more domain-specific and less reusable than quality factors and subfactors because they are specific descriptions of specific system and subsystems. To deal with the large number of criteria and to make them reusable, quality criteria can often be parameterized in the quality models, and specific instances of the parameterized classes of criteria can then be used to produce quality criteria.

- **Quality Threshold**

  A *quality threshold* specifies a minimum level of quality along a quality measurement scale. Thus, the threshold is measured in units of measure based on the quality measure of the quality model for the quality factor or quality subfactors associated with the quality criteria of the quality requirement. For example, throughput performance requirements may specify a minimum acceptable quality threshold of a certain number of transactions per second. Similarly, a reliability requirement may specify a minimum acceptable quality threshold of a certain mean time between failures (MTBF).

  Perhaps the most important thing to remember about quality requirements is that all should have clearly stated quality thresholds. Without quality thresholds, it is not a quality requirement but rather a vague, ambiguous quality goal that is therefore unverifiable. From a system architecture quality assessment standpoint, without quality requirements with associated quality thresholds, it is impossible for

  – system architects to *properly* make engineering tradeoffs between competing quality requirements

[59] Under certain circumstances, a quality criterion may be related to more than one quality subfactor. For example, the quality criteria of defensibility requirements (e.g., safety, security, and survivability requirements) typically address both a defensibility problem subfactor (e.g., harm, danger, defensibility event, or risk) and a defensibility solution subfactor (e.g., prevention, detection, and reaction). Thus, a safety requirement may specify that a system must prevent accidental harm, detect an accident, or react a specific way to the detection of an accident).

- system architects and system architecture assessors to *know* if the system architecture
  - sufficiently supports its derived and allocated quality requirements
  - is therefore good enough
- assessors to unambiguously *determine* if the system architecture should pass the assessment

Appendix D Example Checklists

D.1 Example Subsystem Requirements Meeting Checklist

The following questions can be used as a checklist during the preparation for and performance of a subsystem requirements meeting.

D.1.1 Preparation for the Requirements Review Meeting

The following questions concern preparation for the requirements review:

1. **Method Training**

   Did the subsystem requirements team and the subsystem architecture team receive training in the latest version of the tailored system architecture quality assessment method?

2. **Quality Factors and Quality Subfactors**

   Has the subsystem requirements team identified the relevant quality factors and subfactors?

3. **Preparatory and Presentation Materials**
   - **Were necessary materials provided?**
     - Did the subsystem requirements team and subsystem architecture team provide the necessary preparation and presentation materials to the assessment team?
     - Were these materials complete in the sense of documenting the quality goals and requirements as well as providing sample quality cases?
   - **Was there sufficient lead time?**
     Did the subsystem requirements team and subsystem architecture team provide the preparation and presentation materials to the assessment team with sufficient lead time so that the assessment team members could read these materials prior to the subsystem requirements review meeting?
   - **Was relevant material indicated?**
     Was the relevant information identified (e.g., highlighted or indexed) in a manner that it was easy for the assessment team to find?

4. **Assessment Team Preparation**

   Did members of the assessment team read the preparatory and presentation materials provided by the subsystem requirements team and subsystem architecture team?

5. Meeting Organization

- Have meeting attendees and stakeholders been identified?
- Have the date, time, and location of the meeting been set?
- Has an agenda been developed?
- Have the intended meeting members been invited to the meeting?
- Have other stakeholders been notified of the meeting?

D.1.2 Quality Goals and Requirements

The following questions concern the quality goals and requirements:

1. Quality Goals

For each relevant quality factor and subfactor, did the subsystem requirements team identify and derive a complete set of goals for the subsystem?

2. Quality Requirements

For each relevant quality factor and subfactor, did the subsystem requirements team identify and derive a complete set of quality requirements for the subsystem?

3. Quality of the Quality Requirements

Are the derived and allocated quality requirements correct, complete,[60] consistent (with the associated quality goals), unambiguous, and verifiable?

4. Requirements Trace

Did the requirements team develop a complete requirements trace showing the derivation of the subsystem quality goals and requirements to their sources?

5. Understanding of the Quality Goals and Requirements

Did the architecture team adequately demonstrate their understanding of the quality goals and requirements?

[60] Because QUASAR is an architecture quality assessment method rather than a requirements assessment method, it is *not* intended that the requirements team provide a great deal of evidence to show correctness and completeness. The important thing is to be able to convince the assessment team that the requirements are unambiguous and verifiable so that the assessment team will be able to assess the architecture against these requirements and feel confident that the assessment team and the architecture team will be able to know whether or not the architecture adequately supports these requirements.

D.1.3 Sample Architecture

The following questions concern the sample architecture:

1. **Planned Quality Case Arguments and Evidence**

   Did the architecture team present a representative sample of the kind of quality case arguments and evidence they intend to provide and present during the subsequent architecture assessment meeting?

2. **Quality of Planned Arguments**
   - **Relevant Arguments?**
     Did the architecture team present representative arguments that were relevant in the sense that the arguments were actual architectural decisions (e.g., architectural patterns and mechanisms) rather than merely architectural plans and procedures (i.e., intended ways for future development of the architecture)?
   - **Clear and Compelling?**
     Were the arguments clear and compelling to the assessors?
   - **Rationales?**
     Were rationales provided with each of the architectural decisions?

3. **Quality of Planned Evidence**
   - **Relevant Evidence?**
     Did the architecture team provide relevant evidence (e.g., architecture documents, models, and diagrams that backed up their relevant arguments)? In other words, did the evidence show actual architectural decisions?
   - **Adequate Evidence?**
     Did the evidence back up all of the arguments?
   - **Official Evidence?**
     Is the evidence official project documentation under configuration management as opposed to temporary PowerPoint presentations developed just for the assessment?
   - **Evidence Indicated?**
     Did the architects identify the relevant information (e.g., by highlighting, indexing, or verbally during the presentation) amongst the potentially vast amount of evidentiary documentation provided?

4. **Action Item List**
   - **Action Items Identified?**
     Was the action item list updated based on action items identified during the subsystem requirements review phase?
   - **Assigned with Due Dates?**
     Have actions been assigned with due dates?
   - **Tracked to Completion?**
     Are action items being tracked to completion?
   - **Action Item List Maintained?**
     Is the action item list being properly maintained as action items are completed?

D.1.4 Follow-Through Task

The following questions concern the subsystem requirements review phase follow-through task:

1. **Outbrief given?**

   At the end of the requirements review meeting, did the assessment team develop and present an adequate outbrief to the members of the subsystem requirements team, members of the subsystem architecture team, and other interested stakeholders (e.g., managers and members of the top-level architecture team)?

2. **Meeting Minutes**
   - **Developed?**
     Did the assessment team develop a complete set of minutes of the meeting?
   - **Minutes Quality?**
     Were the minutes reviewed for completeness and factual correctness?
   - **Distributed?**
     Were the minutes distributed to all relevant stakeholders?
   - **Timeliness?**
     Were the minutes developed, reviewed, and distributed within a reasonable time after the meeting?

3. **Lessons Learned**

   Were lessons learned during the requirements review phase captured? Were these lessons learned used to update the assessment method and associated training materials?

D.2 Example Architecture Assessment Checklist

The following questions can be used as a checklist during the preparation for and performance of a subsystem architecture assessment meeting.

D.2.1 Questions Concerning Preparation for the Meeting

The following questions concern preparation for the architecture assessment meeting:

1. **Preparatory and Presentation Materials**
   - **Necessary Materials Provided?**
     - Did the subsystem architecture team provide the necessary preparation and presentation materials to the assessment team?
     - Were these materials complete in the sense of introducing the subsystem, reviewing the derived and allocated quality requirements, introducing the subsystem architecture, and documenting the quality cases?

- **Sufficient Lead Time?**
  Did the subsystem architecture team provide the preparation and presentation materials to the assessment team with sufficient lead time so that the assessment team members could read these materials prior to the architecture assessment meeting?
- **Relevant Material Indicated?**
  Was the relevant information identified (e.g., evidence highlighted or indexed) in a manner that it was easy for the assessment team to find?

2. Assessment Team Preparation

Did members of the assessment team read the preparatory and presentation materials provided by the subsystem architecture team?

3. Requests for Information and Action

- Did members of the assessment team create any RFIs and RFAs based on their reading of the preparation and presentation materials?
- Did these RFIs and RFAs get delivered to the subsystem requirements team with sufficient lead time for them to prepare their responses before the subsystem assessment meeting?

4. Meeting Organization

- Have meeting attendees and stakeholders been identified?
- Have the date, time, and location of the meeting been set?
- Has an agenda been developed?
- Have the intended meeting members been invited to the meeting?
- Have other stakeholders been notified of the meeting?

D.2.2 Initial Presentations

The following questions concern the initial presentations made by the subsystem architecture team during the architecture assessment meeting:

1. System Introduction

- Did the subsystem architecture team provide the assessment team with an adequate introduction to the subsystem?
- Did the presenters address the primary purpose of the subsystem, where it fits into the overall system architecture, its context in terms of the externals and other subsystems with which it interoperates, and its primary functions?

2. Requirements Review

- Did the subsystem architecture team review the architecturally significant goals and requirements that have been derived and allocated to the subsystem?
- Did the presenters identify the relative priorities of these goals and requirement types (and requirements where appropriate)?

3. Architecture Introduction

- Did the subsystem architecture team introduce their architecture to the members of the assessment team?

- Did they cover its major architectural components and their objectives?
- Did they cover the major relationships between these architectural components?
- Did they provide an overview of their most important architectural decisions (e.g., choice of architectural styles, patterns, and mechanisms) and their associated rationales?
- Did they describe the major engineering tradeoffs that they made between conflicting quality factors?

4. Quality Cases

- Did the subsystem architecture team present their quality cases to the assessment team?
- Did they cover all of the relevant important quality factors and quality subfactors given the time constraints of the meeting?
  - *Claims?*
    - Did the subsystem architecture team make appropriate claims given the quality goals and requirements?
    - Did these claims include requirements, or only goals?
    - Were the requirements claims unambiguous in terms of specific, quantified quality requirements?
  - *Arguments?*
    - Did the subsystem architecture team provide arguments as to why the assessment team should believe their claims?
    - Where these arguments clear and compelling?
    - Were the arguments stated in terms of specific architectural decisions and associated rationales?
    - Did the architects use standard architectural styles, patterns, and mechanisms?[61]
    - Were their decisions appropriate given the architecturally significant requirements and other architectural constraints?
    - Were the rationales credible?
  - *Evidence?*
    - Was the evidence in the form of official project documents or assessment team witnessed demonstrations?
    - Was relevant information identified (e.g., evidence highlighted or indexed) in a manner that it was easy for the assessment team to find?
    - If asked for more specific evidence, could the architects easily find and display it?

[61] This checklist can be augmented with standard quality-factor specific architecture decisions, but experience has shown that it is usually best left to separate training material, as its inclusion makes the checklist too long and unwieldy for use during assessment meetings.

5. **Probe Architecture**
   - Did the assessment team adequately probe the architecture to identify weaknesses and defects not brought out in the subsystem architecture team's quality cases?
   - Were the subsystem architecture team members able to satisfactorily answer all assessment team questions?
   - Were deep dives needed in any areas of the subsystem architecture?
   - Were scenarios needed to exercise the architecture?
6. **Action Item List**
   - **Action Items Identified?**
     Was the action item list updated based on action items identified during the subsystem architecture assessment phase?
   - **Assigned with Due Dates?**
     Have actions been assigned with due dates?
   - **Tracked to Completion?**
     Are action items being tracked to completion?
   - **Action Item List Maintained?**
     Is the action item list being properly maintained as action items are completed?

D.2.3 Follow-Through Task

The following questions concern the subsystem architecture assessment phase follow-through task:

1. **Outbrief given?**

   At the end of the subsystem architecture assessment meeting, did the assessment team develop and present an adequate outbrief to the members of the subsystem architecture team, and other interested stakeholders (e.g., managers and members of the top-level architecture team)?
2. **Meeting Minutes**
   - **Developed?**
     Did the assessment team develop a complete assessment report?
   - **Minutes Quality?**
     Was the assessment report reviewed for completeness and factual correctness?
   - **Distributed?**
     Was the assessment report distributed to all relevant stakeholders?
   - **Timeliness?**
     Were the assessment report developed, reviewed, and distributed within a reasonable time after the meeting?
3. **Lessons Learned**
   - Were lessons learned during the subsystem architecture assessment phase captured?
   - Were these lessons learned used to update the assessment method and associated training materials?

Appendix E Example Quality Cases

For each relevant quality factor (e.g., availability), the associated quality case consists of the following three sets of related information:

1. **Claims**
   A *claim* is an architect's assertion that his or her architecture adequately supports the associated type of quality goal or requirements (e.g., availability requirements). These claims should list[62] and/or summarize the specific, relevant quality requirements.

2. **Arguments**
   An *argument* is one of an architect's reasons why the assessors should believe his or her associated claims. Essentially, the architects present clear and compelling arguments listing the specific architectural decisions that they have made (and documented) to ensure that the system will achieve a sufficient level of the associated type of quality (e.g., availability).

3. **Evidence**
   Evidence is the documentation that is provided by the architects to convince the assessors of the validity of the arguments. Typical examples of appropriate evidence consists of official project documentation including architectural diagrams, architectural models, architectural documents, architectural white papers, and architectural training materials. Documentation of architecture drivers (e.g., architecturally significant requirements, associated use cases) and the architects' intent (e.g., architectural plans, architectural procedures, or architecture team charters) is *not* adequate because it documents neither the current state of the architecture nor the actual architectural decisions. Valid evidence can also consist of demonstrations during which the assessors directly witness the incorporation of architectural decisions; for example, the assessors can observe the incorporation of redundant hardware into an executable hardware prototype during a tour of a development or testing laboratory.

The following sections of this appendix contain architectural quality cases that are general examples of the kind of claims, arguments, and evidence that architects might present to assessors. These examples have been highly generalized and are not taken from any particular

[62] It is insufficient to merely list requirements identifiers (e.g., numbers) because this makes it very difficult for assessors to understand the relevant requirements. If requirements management tools are used to assign specific quality factors as attributes to the associated quality requirements and to trace such requirements to architectural elements, then it becomes practical to select all quality requirements of a specific type allocated to a specific subsystem and therefore automatically generate the associated claims (assuming that the subsystem adequately supports its allocated quality factors).

acquisition or development project. Any relationship to any particular project is purely coincidental.

E.1 Example Interoperability Cases

Interoperability is the degree to which a system (or subsystem) operates effectively with specified [types of] external systems by successfully providing services and data to those systems and using services and data provided by those systems.

Interoperability is decomposed into the following subfactors:

- **Physical Interoperability**
  Physical interoperability is the degree to which the system or subsystem physically connects with specified [types of] interfaces with specified [types of] external systems. Physical interoperability includes matching the
  - Electrical Connections – electrical plug type, power rating, number and configurations of prongs, male versus female connection, and so on
  - Electronic Connections – number and configuration of pins, male versus female connection, and so on
  - Physical Connections – size and shape of surfaces as well as the number, type, orientation, and size of physical connectors such as bolts and screws
  - Power Connections – cable, chain, and hose
- **Energy Interoperability**
  Energy interoperability is the degree to which the system or subsystem correctly uses the energy types and levels of the specified [types of] external systems. Energy interoperability includes
  - Hydraulics – fluid type, maximum and minimum pressures, and so on
  - Mechanical Linkages – linkage type (e.g., belt, cable, or chain) and average and maximum force
  - Wired Communication – proper logic voltages, frequency, and amperage of electricity such as logic low level (0–1 volts), logic high level (3.5–5.0 volts), 40 megahertz (MHz), and 100 milliamps
  - Wired Power – proper voltages including ranges, AC (alternating current) frequency including ranges, and maximum amperage of electricity such as 120 volts AC or 270 VDC (volts of direct current) (250–280 VDC), 60 hertz, and 100 milliamps or 150 kilowatts maximum
  - Wireless Communication
    Wireless communication energy operability includes
    - proper electromagnetic frequency (e.g., proper spectrum of radio waves, microwaves, or visible light) such as 1850–1990 MHz microwaves for cellular telephone transmission or C-band for satellite transmission
    - broadcast or laser
    - minimum/maximum signal strength

- **Protocol Interoperability**
  Protocol interoperability is the degree to which the system or subsystem correctly uses the interface protocols of the specified [types of] external systems. Protocol interoperability includes compatibility of protocols at the following layers:
  - Physical Layer Protocols – Layer 1 protocols such as Integrated Services Digital Network (ISDN) and RS-232
  - Data Link Layer Protocols – Layer 2 protocols such as Ethernet, Fiber Distributed Data Interface (FDDI), and Point-to-Point Protocol (PPP)
  - Network Layer Protocols – Layer 3 protocols such as Internet Control Message Protocol (ICMP) and Internet Protocol (IP)
  - Transport Layer Protocols – Layer 4 protocols such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)
  - Session Layer Protocols – Layer 5 protocols such as Network File System (NFS)
  - Presentation Layer Protocols – Layer 6 protocols such as American Standard Code for Information Interchange (ASCII), Moving Picture Experts Group (MPEG), and Secure Socket Layer (SSL)
  - Application Layer Protocols – Layer 7 protocols such as Domain Name Service (DNS), File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), and Hypertext Transfer Protocol Secure (HTTPS)
- **Syntax Interoperability**
  Syntax interoperability is the degree to which the system or subsystem correctly communicates data having the correct syntax (e.g., data types such as text, integer, date, and money including associated attributes, ranges, and default values) with specified [types of] external systems.
- **Semantics Interoperability**
  Semantics interoperability is the degree to which the system or subsystem communicates requests and data via specified [types of] interfaces of the specified [types of] external systems in a manner that both systems interpret the syntax in a single standard way to gain the same meanings. Semantic interoperability includes such things as units of measure (e.g., differentiating between English and metric units or differentiating between Canadian and United States currency).

Sections E.1.1 and E.1.2 include example interoperability cases for the physical interoperability and protocol interoperability quality subfactors of interoperability, respectively.

As illustrated in Figure 23, an interoperability case consists of one or more interoperability claims, the belief in which is justified by one or more interoperability arguments supported by one or more pieces of interoperability evidence.

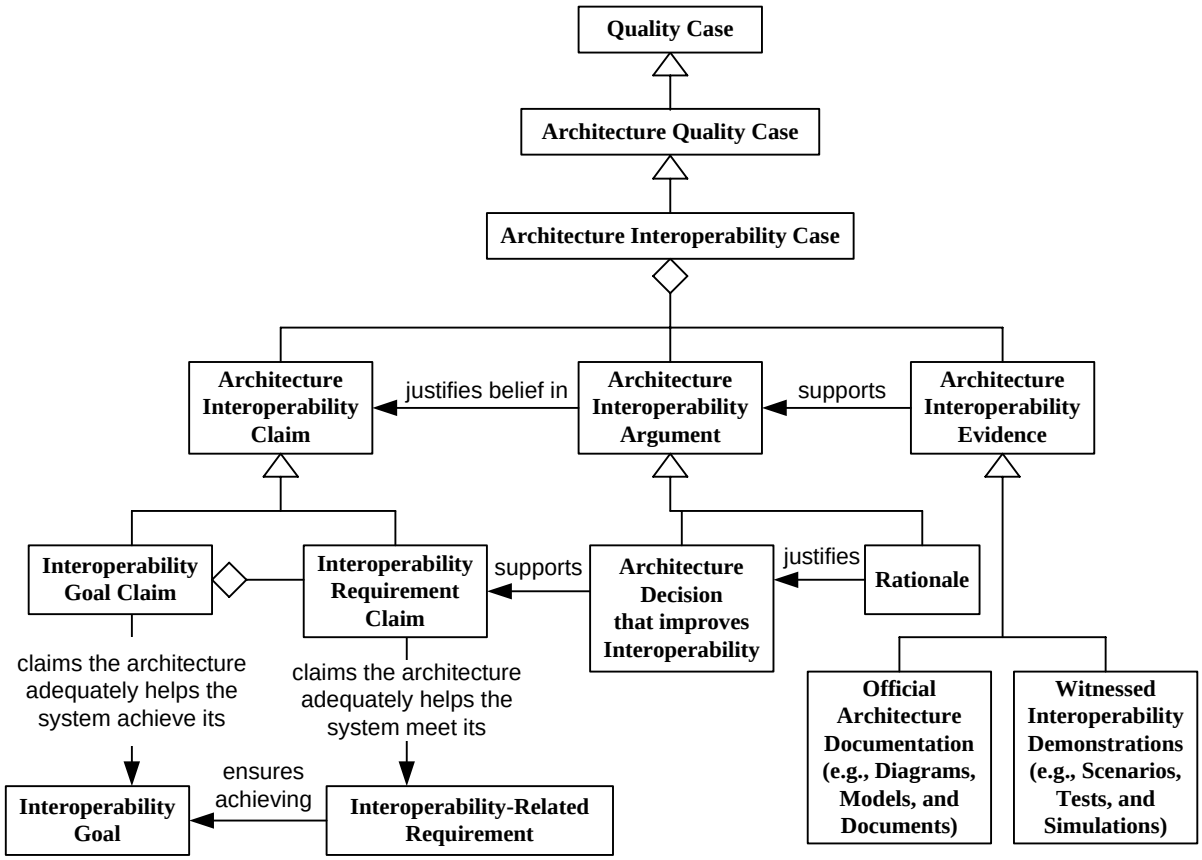

Figure 23: Components of an Architecture Interoperability Case

Figure 24 is an example interoperability quality case diagram showing the relationships between the different types of interoperability claims, arguments, and evidence.

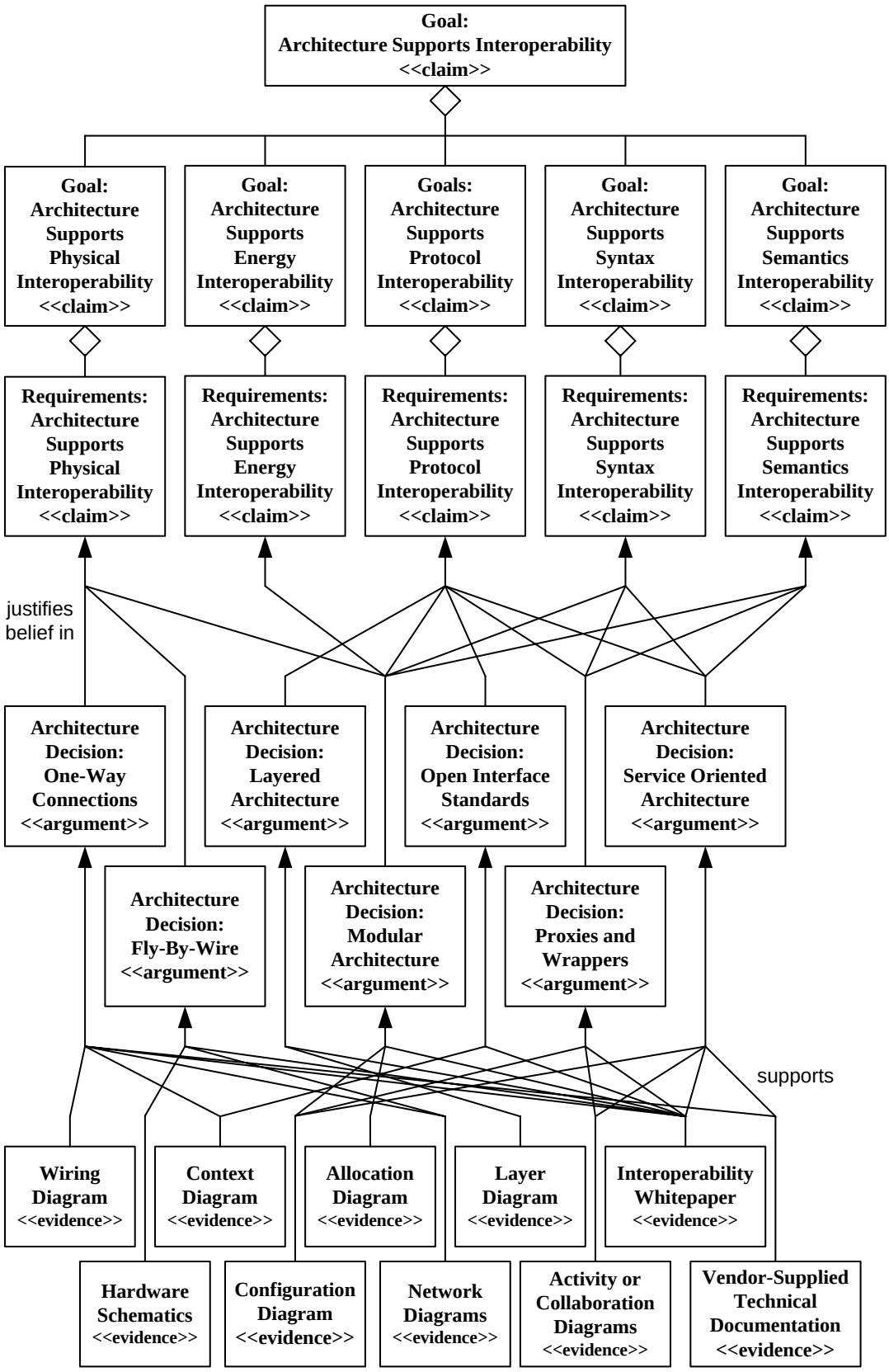

Figure 24: Example Interoperability Quality Case Diagram

E.1.1 Example Physical Interoperability Case

As an interoperability subfactor, *physical interoperability* is the degree to which the system or subsystem physically connects with specified [types of] interfaces with specified [types of] external systems. Physical interoperability includes matching the

- **Electrical Connections** – electrical plug type, power rating, number and configurations of prongs, male versus female connection, and so on
- **Electronic Connections** – number and configuration of pins, male versus female connection, and so on
- **Physical Connections** – size and shape of surfaces as well as the number, type, orientation, and size of physical connectors such as bolts and screws
- **Power Connections** – cable, chain, and hose

The example physical interoperability case is made up of the claims, arguments, and evidence presented in Sections E.1.1.1, E.1.1.2, and E.1.1.3, respectively.

E.1.1.1 Example Physical Interoperability Claims

The example physical interoperability case includes the following example claims:

- **Goals**
  - **Architecture Supports Physical Interoperability Goal**
    Claim: The architecture adequately supports the system or subsystem's ability to physically interoperate with external systems.
- **Requirements**
  - **Architecture Supports Physical Interoperability Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived physical interoperability requirements that have been allocated to it:
    - "Subsystem X shall have dimensions 42 cm by 55 cm by 100 cm and attach to system Y with attachment types and locations as indicated in table Z."

E.1.1.2 Example Physical Interoperability Arguments

The example physical interoperability case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **One-Way Connections**
  Architectural Decision: The system or subsystem architecture uses unique asymmetric electric, electronic, and physical connections between it and external systems.
  Rationale: By using direction and conformation to ensure that connections can only work one way, it becomes impossible to make incorrect connections.

- **Fly-By-Wire**
  Architectural Decision: The system or subsystem architecture uses electronic rather than hydraulic connections.
  Rationale: Electronic connections produce higher reliability.

- **Modularity with Minimal Cohesion**
  Architectural Decision: The system or subsystem architecture is highly modular with minimal cohesion between components.
  Rationale: High modularity allows architectural elements (modules) to be allocated to handle interoperation with external systems. Minimizing cohesion decouples these interfacing modules from the rest of the architecture.

E.1.1.3 Example Physical Interoperability Evidence

The example physical interoperability case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Context Diagram**
  The context diagram shows the external systems or hardware components with which the system or subsystem physically interoperates.

- **Configuration Diagram**
  The configuration diagram shows the hardware components that physically interoperate.

- **Hardware Diagrams**
  Hardware diagrams show the physical connections between hardware components that physically interoperate.

- **Interoperability White Paper**
  The interoperability white paper describes how physical interoperability is implemented including how it is supported by the architecture.

- **Wiring Diagrams**
  Wiring diagrams show how physical components are physically connected together.

E.1.2 Example Protocol Interoperability Case

As an interoperability subfactor, *protocol interoperability* is the degree to which the system or subsystem correctly uses the interface protocols of the specified [types of] external systems.

The example protocol interoperability case is made up of the claims, arguments, and evidence presented in Sections E.1.2.1, E.1.2.2, and E.1.2.3, respectively.

E.1.2.1 Example Protocol Interoperability Claims

The example protocol interoperability case includes the following example claims:

- **Goals**
  - **Architecture Supports Protocol Interoperability Goals**
    Claim: The architecture adequately supports the system or subsystem's ability to achieve the following derived protocol goals that have been allocated to it:
    - "The system or subsystem shall correctly use the interface protocols of all relevant external systems."
    - "The system or subsystem shall use open interface standards (i.e., industry standard protocols) when communicating with external systems."
- **Requirements**
  - **Architecture Supports Protocol Interoperability Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived protocol interoperability requirements that have been allocated to it:
    - "The system or subsystem shall use open interface standards (i.e., industry standard protocols) when communicating with external systems across all key interfaces identified in document X."
    - "The system or subsystem shall use the Ethernet over RS-232 for communication across interface X with external system Y."
    - "The system shall use HTTPS for communicating securely when performing function X across interface Y with external system Z."

E.1.2.2 Example Protocol Interoperability Arguments

The example protocol interoperability case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Layered Architecture**
  Architectural Decision: The system architecture will consist of the following horizontal layers including:
  - Interface Layer, which contains software for communication with human users and external systems
  - Application Process Layer, which contains and hides application-level processes and transactions
  - Business Layer, which contains and hides business rules and business objects
  - Persistence Layer, which contains and hides system-specific databases
  - Legacy Layer, which provides proxies for legacy applications and databases

  Rationale: A layered architecture contains, logically groups, separates, and hides different parts of the system. The interface layer supports interoperability with external systems, whereas the legacy layer supports interoperability with legacy systems.
- **Modular Architecture**
  Architectural Decision: The system architecture will consist of a relatively large number of small, cohesive, modular subsystems and sub-subsystems with their implementations hidden behind well-defined interfaces.

Rationale: Using a highly modular architecture will support the creation of modules that act as local proxies for the external systems with which the system must interoperate. As such, they will act as wrappers that hide the external interfaces.

- **Proxies and Wrappers**
  Architectural Decision: For each external client and server system with which the system must interoperate, the system architecture will include one module that act as a proxy for that external system. The proxy model will act as a wrapper hiding the interfaces to the external system.

  Rationale: Proxies localize and wrap all information about the external interfaces, making it easier to change them.

- **Service-Oriented Architecture (SOA)**
  Architectural Decision: The system architecture will be based on the SOA pattern to provide software as Web services over the Internet. It will use standard Web Services Interoperability Organization's[63] (WS-I) Basic Profile 1.0 consisting of the Extensible Markup Language (XML) Schema 1.0 and associated protocols: Simple Object Access Protocol (SOAP) 1.1, Web Services Description Language (WSDL) 1.1, and Universal Description, Discovery, and Integration (UDDI) 2.0. The system will register (i.e., publish) its services with a Web services directory so that clients of the system can bind and invoke (i.e., execute) the services.

  Rationale: Interoperability and portability are the primary principles underlying the SOA pattern. Use of standard languages and protocols enable interoperability between heterogeneous systems. Web services are software components with well-defined interfaces that hide their implementation technologies. Web services are self-contained, loosely coupled, and dynamically discovered.

E.1.2.3 Example Protocol Interoperability Evidence

The example protocol interoperability case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Context Diagram**
  The context diagram shows the context of the system in terms of the external systems with which it must interoperate. It therefore identifies and names the interfaces to these external systems. It potentially provides evidence of the open interface standards used on these external interfaces by identifying the type of interface and listing its protocol.

- **Configuration Diagram**
  The configuration diagram shows the aggregation hierarchy of the system in terms of its subsystems, their subsystems, and so on. It therefore provides evidence of the
  - modularity of the system's architecture
  - existence of the architectural components that

[63] For more information, visit http://www.ws-i.org/.

  - act as proxies for (or wrappers around) external systems
  - provide Web services

- **Allocation Diagram**
  The allocation diagram shows the allocation of data and software components to the hardware components of the system architecture. It therefore provides some evidence of the modularity of the system architecture.

- **Layer Diagram**
  The layer diagram identifies the horizontal layers of software that make up the system and implicitly shows the limitation of internal interfaces between these layers. It therefore provides clear and direct evidence that a layered architecture pattern was used.

- **Activity/Collaboration Diagram (treating subsystems as objects)**
  Activity and collaboration diagrams show the interactions between the classes that make up the system. When used as architecture diagrams, the classes represent subsystems, sub-subsystems, and so on. These diagrams therefore provide evidence of the
  - names and locations of proxies and wrappers
  - services of the SOA and how they are accessed and used

- **Interoperability White Paper**
  The interoperability white paper documents architectural styles, patterns, and mechanisms used to achieve protocol interoperability.

- **Vendor-Supplied Technical Documentation**
  Vendor-supplied technical documentation documents commercial off-the-shelf (COTS) product support for standard SOA protocols including XML, SOAP, WSDL, and UDDI.

E.2 Example Performance Cases

Performance is the degree to which a system or subsystem operates within its designated temporal constraints. The quality subfactors of performance include jitter, latency, response time, schedulability, and throughput.

Sections E.2.1 and E.2.2 include example performance cases for the jitter and latency quality subfactors of performance, respectively.

As illustrated in Figure 25, a performance case consists of one or more performance claims, the belief in which is justified by one or more performance arguments supported by one or more pieces of performance evidence.

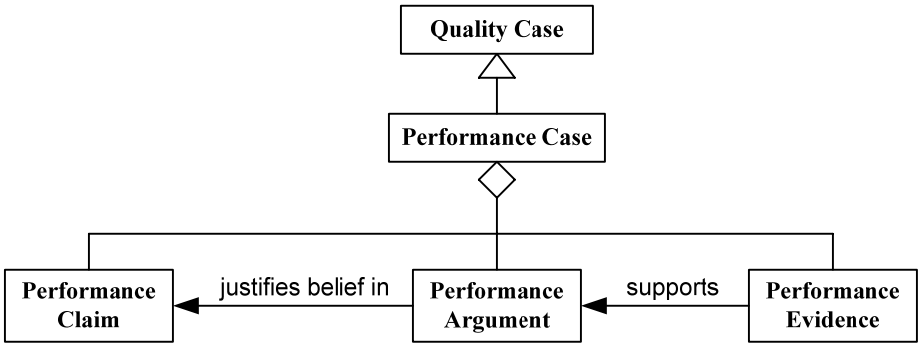

Figure 25: Components of a Performance Case

Figure 26 is an example performance quality case diagram showing the relationships between the different types of performance claims, arguments, and evidence.

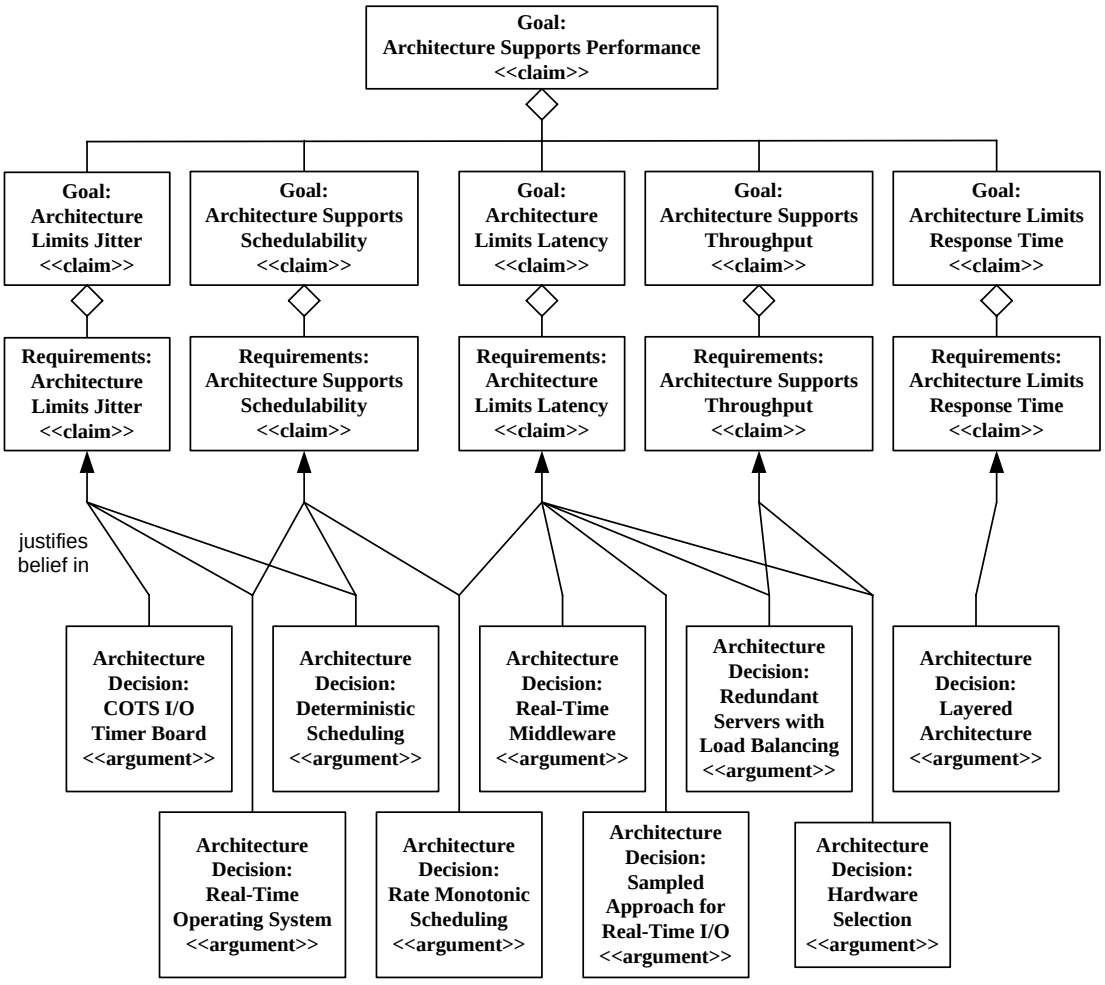

Figure 26: Example Performance Quality Case Diagram

E.2.1 Example Jitter Performance Case

As a performance subfactor, *jitter* is the degree to which the variability of the time intervals between controlled periodic actions remains within its designated constraints.

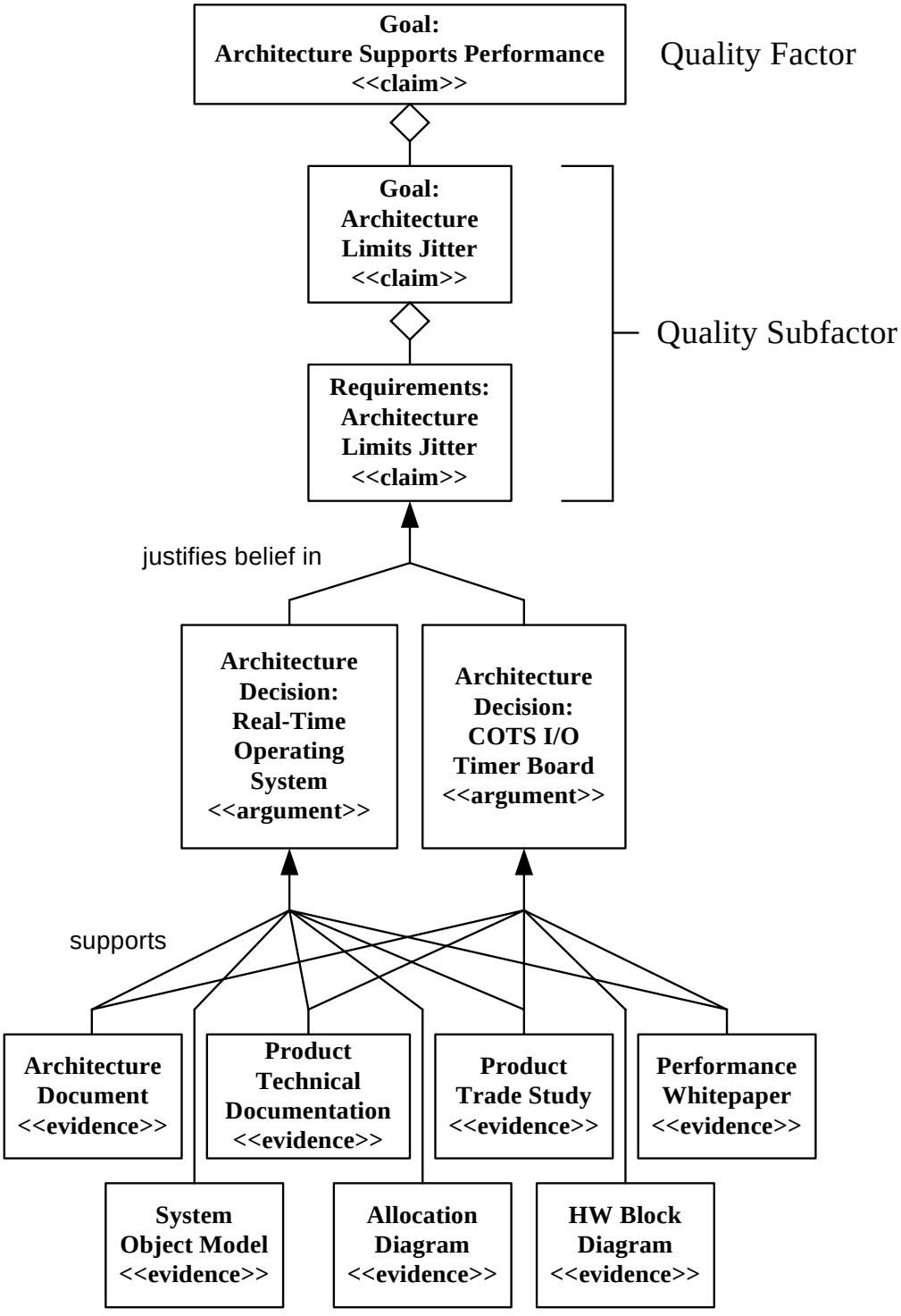

Figure 27: Example Jitter Quality Case Diagram

The example jitter performance case is made up of the claims, arguments, and evidence presented in Sections E.2.1.1, E.2.1.2, and E.2.1.3, respectively.

E.2.1.1 Example Jitter Claims

The example jitter performance case includes the following example claims:

- **Goals**
  - **Architecture Supports Jitter Goal**
    Claim: The architecture adequately supports the system or subsystem's ability to constrain jitter to acceptable limits.
- **Requirements**
  - **Architecture Supports Jitter Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived jitter requirement that has been allocated to it:
    - "The robotic surgery system shall control the blade actuator input/output (I/O) with a jitter of less than one tenth of a millisecond."

E.2.1.2 Example Jitter Arguments

The example jitter performance case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Real-Time Operating System (OS)**
  Architectural Decision: The system architecture incorporates a proprietary COTS real-time operating system from a well-established vendor.

  Rationale: The real-time OS comes with the needed device drivers and with design tools that engineers have been trained to use. It also comes with deterministic, real-time resource management including jitter management on I/O.
- **COTS HW I/O Timer Board**
  Architectural Decision: The system architecture incorporates a COTS hardware I/O board with sufficient channels, timer resolutions, and on-board buffer size to handle the required traffic jitter requirement.

  Rationale: The server processor board does not have adequate time resolution to handle the jitter requirements.

E.2.1.3 Example Jitter Evidence

The example jitter performance case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Architecture Document**
  The architecture document provides a high-level overview the system architecture's incorporation of a
  - real-time operating system
  - COTS I/O Timer Board
- **Product Technical Documentation**
  The vendors' product technical documentation documents the
  - real-time operating system technical characteristics
  - COTS I/O Timer Board technical characteristics
- **Product Trade Studies**
  Product trade studies document the
  - real-time operating system technical characteristics (operating system trade study)
  - COTS I/O Timer Board technical characteristics (timer board trade study)
- **Performance White Paper**
  The performance white paper provides a detailed analysis of the system architecture's incorporation of (and rationale for)
  - real-time operating system
  - COTS I/O Timer Board
- **System Object Model**
  The system object model identifies the real-time operating system and possibly also the COTS I/O Timer Board as system objects and the relationships between them and other system objects.
- **Allocation Diagram**
  The allocation diagrams document how the real-time operating system is allocated onto the selected hardware.
- **Hardware Block Diagram**
  The system hardware block diagrams show how the COTS I/O Timer Board is connected to other hardware components.

E.2.2 Example Latency Performance Case

As a performance subfactor, *latency* is the degree that the time that the system or subsystem takes to execute specific tasks (e.g., system operations and use case paths) from end-to-end is within acceptable limits.

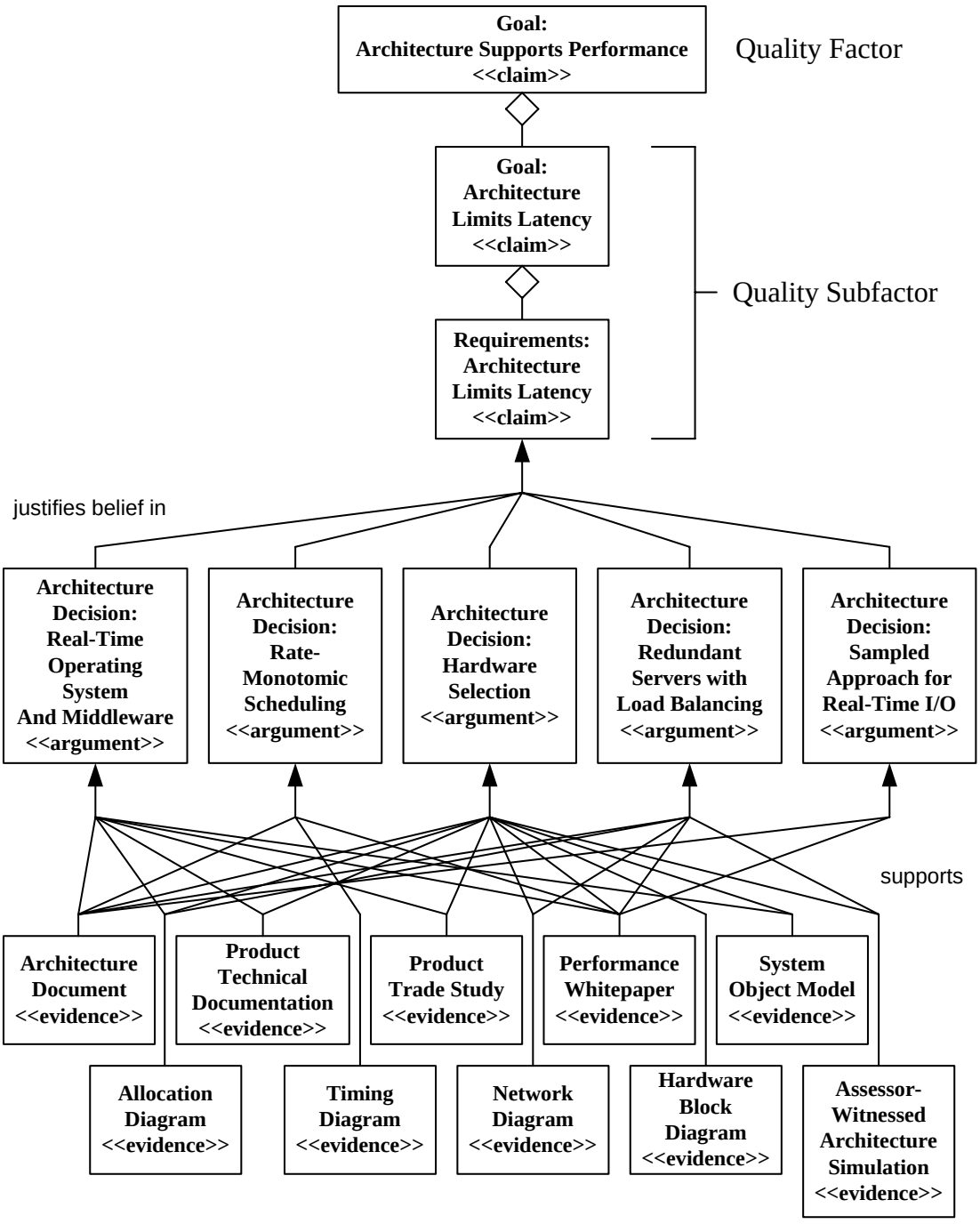

Figure 28: Example Latency Quality Case Diagram

The example latency performance case is made up of the claims, arguments, and evidence presented in Sections E.2.2.1, E.2.2.2, and E.2.2.3, respectively.

E.2.2.1 Example Latency Claims

The example latency performance case includes the following example claims:

- **Goals**
  - **Architecture Supports Latency Goal**
    Claim: The architecture adequately supports the system or subsystem's ability to constrain latency to acceptable limits.
- **Requirements**
  - **Architecture Supports Latency Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived latency requirements that have been allocated to it:
    - *Hard Deadline Requirement*
      "The Vat Subsystem shall send either a Vat Sensor Values message or a Sensor Data Unavailable message to the Production Control Subsystem once every 50 milliseconds."
    - *Soft Deadline Requirement*
      "The Vat Subsystem shall send Vat Sensor Values messages and Sensor Data Unavailable messages to the Production Control Subsystem in such a manner that Vat Sensor Value messages are not separated by more than four consecutive Sensor Data Unavailable messages."

E.2.2.2 Example Latency Arguments

The example latency performance case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Real-Time Operating System and Middleware**
  Architectural Decision: The system architecture incorporates a proprietary COTS real-time operating system from a well-established vendor.

  Rationale: The real-time operating system and middleware enables the use of rate-monotonic scheduling, which will be used to support the meeting of the latency requirements (see next).

- **Rate-Monotonic Scheduling (RMS)**
  Architectural Decision: The system architecture requires that all subsystems having real-time latency requirements incorporate COTS hardware, a real-time OS, and real-time middleware that either directly support RMS or permit the building of "scheduling" wrappers with sufficiently small schedulability penalties.

  Rationale: The architecture's use of RMS not only helps the system to meet its latency requirements, it also supports end-to-end schedulability analysis to show during early development that these requirements can be met. RMS was also chosen because
  - All real-time traffic is either periodic or can be buffered and handled by periodic services, a prerequisite for RMS.
  - RMS is supported by most hardware and software real-time standards and standards-compliant COTS products.

- RMS is a mathematically sound method guaranteeing schedulability under the system's conditions.
- The system traffic profiles can be handled by RMS.

- **Hardware Selection**
  Architectural Decision: The system architecture incorporates N_1 hardware clients of type HC_1, N_2 hardware servers of type HS_2, N_3 network devices of type ND_3, N_4 storage devices of type SD_4, and N_5 networks of type N_5.[64]

  Rationale: This selection of hardware devices provides the hardware capacity needed to achieve latency requirements under current workloads as well as providing adequate capacity to continue to meet latency requirements under future planned growth. This was determined in an iterative manner using schedulability analysis in light of current estimates of subsystem workloads and communication patterns.

- **Software and Communications Allocation**
  Architectural Decision: The system architecture allocates process P1 to node N1 and processes P2 and P3 to note N2, etc. The system architecture constrains node-to-node communication to message traffic pattern MTP1.

  Rationale: This allocation of processors to nodes keeps node workloads at shared resources within their scheduling constraints. This constraint on communication traffic avoids communication bottle-necks, lowers message hop count by minimizing indirections, and thereby keeps the sum of delays incurred at shared resources low enough to meet end-to-end latency requirements.

- **Redundant Servers with Load-Balancing**
  Architectural Decision: The system architecture incorporates load balancing of redundant servers.

  Rationale: Load-balancing helps to minimize scheduling bottlenecks at nodes, which in turn helps the system meet its latency requirements.

- **Sampled Approach for Real-Time I/O**
  Architectural Decision: The system architecture incorporates a sampled approach for all real-time I/O between its subsystems.

  Rationale: Using a sampled approach (combined with the production of software-supporting fault tolerance) facilitates the system meeting its latency requirements even if an upstream node occasionally misses a deadline. This architectural approach is made possible because the subsystems have well-synchronized clocks with the required timing resolution.

[64] In real life, the architects would provide actual hardware information including such details as number of I/O channels and network bandwidth. Workloads are based on projected usage and schedulability analysis.

E.2.2.3 Example Latency Evidence

The example latency case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Architecture Document**
  The architecture document provides a high-level overview the system architecture's incorporation of
  - real-time operating system and middleware
  - rate-monotonic scheduling
  - the selected hardware
  - redundant servers with load balancing
  - a sampled approach for real-time I/O
- **Product Technical Documentation**
  The vendors' product technical documentation documents the
  - real-time operating system and middleware used including its technical characteristics
  - selected hardware including servers, storage, and network devices
- **Product Trade Studies**
  Product trade studies document the
  - real-time operating system and middleware used including its technical characteristics (software trade study)
  - selected hardware including servers, storage, and network devices (server trade study, storage trade study, and network trade study, respectively)
- **Performance White Paper**
  The performance white paper provides a detailed analysis of the system architecture's incorporation of (and rationale for)
  - real-time operating system and middleware
  - rate-monotonic scheduling
  - the selected hardware
  - redundant servers with load balancing
  - a sampled approach for real-time I/O
- **System Object Model**
  The system object model identifies the real-time operating system, the real-time middleware, and the hardware components as system objects and the relationships between them.
- **Allocation Diagram**
  The allocation diagrams document how the real-time operating system and middleware are allocated onto the selected hardware including the redundant servers.
- **Hardware Block Diagram**
  The system hardware block diagram shows how the selected hardware components are connected including the redundant servers used for load balancing.

- **Network Diagram**
  The system network diagram shows how the selected hardware components are networked including the redundant servers used for load balancing.
- **Timing Diagram**
  The timing diagram documents how rate-monotonic scheduling is incorporated into the deterministic scheduling of frames.
- **Assessor-Witnessed Architecture Simulation**
  The assessors witness a simulation of the executable architecture demonstrating that certain latency requirements are met under specific circumstances.

E.3 Example Security Cases

Security is the degree to which

- a system or subsystem prevents or reduces in probability or severity, detects, reacts to, and adapts to
  - malicious (i.e., unplanned and unintended but not necessarily unexpected by legitimate stakeholders) harm[65] to valuable assets caused by attackers
  - security events (i.e., attacks and probes)
  - vulnerability to threats (i.e., existence of attackers with means, motives, and opportunities)
- security risks associated with the system or subsystem are acceptably low

Note that the architecture of a system or subsystem may need to support requirements for the following subclasses of security:

- **Communications Security (COMSEC)**
  COMSEC is the degree to which communications are protected from attack.
- **Computer Security (COMPUSEC)**
  COMPUSEC is the degree to which computers are protected from attack.
- **Emissions Security (EMSEC or Tempest)**
  EMSEC is the degree to which systems do not emit radiation that is subject to attack.
- **Information Security (INFOSEC)**
  INFOSEC is the degree to which stored and manipulated data are protected from attack.

[65] For example, information security is technically concerned with preventing and detecting the *malicious* disclosure of sensitive identities, data, and communications during an attack, whereas safety is concerned with preventing and detecting their *accidental* disclosure during an accident. Note that both attacks and accidents can result in the same harm to valuable assets (i.e., to individuals, organizations, and their sensitive data) and that the architects can give the same arguments as part of both safety and security cases because they can use the same architectural mechanisms (controls) as both safeguards and countermeasures.

- **Network Security (NETSEC)**
  NETSEC is the degree to which networks are protected from attack.
- **Operations Security (OPSEC)**
  OPSEC is the degree to which system operations are protected from attack.
- **Personal Security (PERSEC)**
  PERSEC is the degree to which personnel are protected from attack.
- **Physical Security (PHYSEC)**
  PHYSEC is the degree to which systems, data centers, and other facilities are protected from physical attack.

As illustrated in Figure 29, a security case is a kind of quality case that consists of the following three components:

1. **Security Claims**
   Assertions made by the architects that the architecture of the system or subsystem being assessed sufficiently supports the achievement of its allocated and derived *security-related goals and requirements*
2. **Security Arguments**
   Adequate clear and compelling reasons to believe claims consisting of a listing of the architects' relevant *architectural decisions* (and associated *rationales*) that were made to ensure that the architecture sufficiently supports the achievement of its allocated security-related requirements claim
3. **Security Evidence**
   Sufficient evidence backing up the architects' arguments consisting of official documentation clearly indicating the architects' relevant architectural decisions

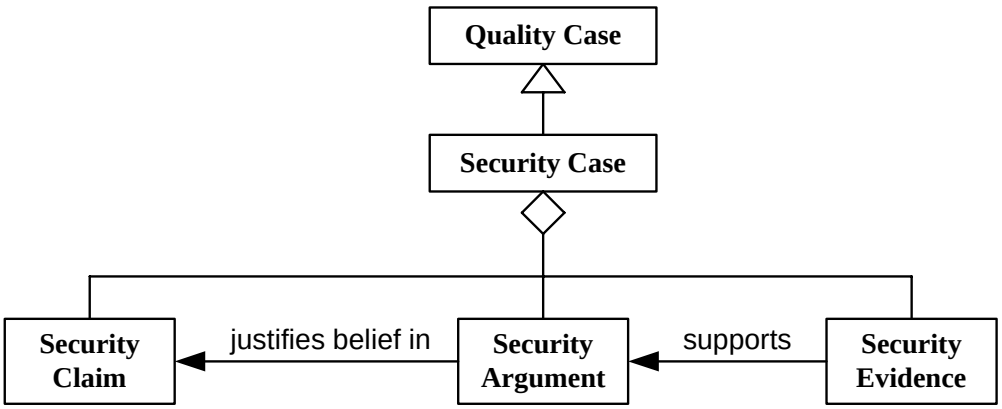

Figure 29: Components of a Security Case

A concern of presenting a security case is that the security claims (i.e., meeting security-related requirements and constraints), arguments (aka, countermeasures), and evidence (e.g., documentation of countermeasures) may itself be highly sensitive (e.g., classified or highly

proprietary). Thus for example, only those members of the assessment team with the proper security clearances may be allowed to examine the security perspectives of the architecture. Although the task of assessing the security aspects of the architecture may be delegated to a security team that is responsible for verifying the system's security, it is important that they include people adequately trained in system architecture and properly assess the system architecture early during the development process.

Security claims are stated in terms the relevant security-related requirements allocated to the system or subsystem, the architecture of which is being assessed. These requirements include security requirements, security-significant requirements (i.e., other requirements having significant security ramifications), requirements of security subsystems, and security constraints.[66] Security requirements in turn specify that the system or subsystem shall provide at least a minimum acceptable amount of security in terms of preventing, detecting, reacting to, and adapting to malicious harm to valuable assets,[67] the occurrence of security events (probes and attacks), threats, and security risks.

Sections E.3.1, E.3.2, E.3.3, and E.3.4 include example security cases for the access control, integrity, privacy, and security auditing subfactors of security, respectively.

E.3.1 Example Access Control Security Case

Access control is the degree to which a system or subsystem controls access by its externals (e.g., human users, external software applications, and external systems) to its data and software components. Access control consists of

- **Identification**
  Identification is the degree to which the system or subsystem establishes the claimed identities of externals (e.g., people, roles, systems) before allowing them to request and receive services (e.g., perform functions, obtain data).
- **Authentication**
  Authentication is the degree to which the system or subsystem verifies the claimed identities of externals before allowing them to request and receive services (e.g., perform functions, obtain data).

[66] A security constraint specifies a countermeasure as a requirement. Requirements engineers must take care not to tie the security architect's hands by unnecessarily specifying a countermeasure (e.g., mandatory use of user IDs and passwords) instead of the underlying true requirement (e.g., that the system must adequately identify and authenticate its users).

[67] Valuable assets include people, property (e.g., data, hardware, software, and money), the environment, and services. Examples of harm to these assets include injury to people, access to or corruption of sensitive data, theft or corruption of hardware, infection of software, theft of money, and theft or denial of services.

- **Authorization**
  Authorization is the degree to which a system or subsystem properly grants and enforces access and usage privileges of authenticated externals.

Figure 30 is an example quality case diagram summarizing the claims, arguments, and evidence composing the example access control security case.

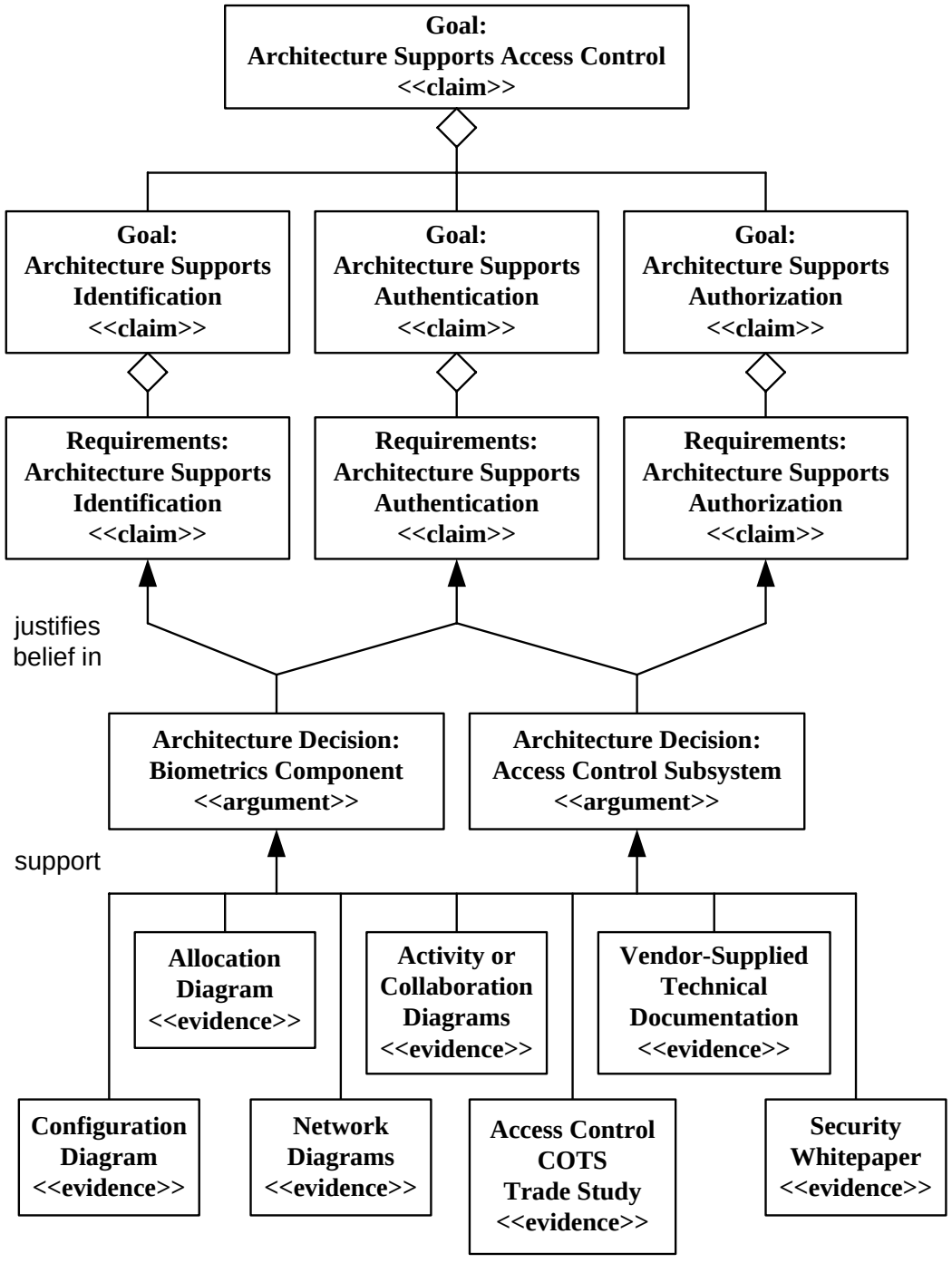

Figure 30: Example Access Control Quality Case Diagram

The example access control security case is made up of the claims, arguments, and evidence presented in Sections E.3.1.1, E.3.1.2, and E.3.1.3, respectively.

E.3.1.1 Example Access Control Claims

The example access control security case includes the following example claims:

- **Goals**
  - **Architecture Supports Access Control**
    Claim: The architecture adequately supports the system or subsystem's ability to grant and restrict usage of sensitive services and access to sensitive data.
    - **Architecture Supports Identification**
      Claim: The architecture adequately supports the system or subsystem's ability to identify users (i.e., human, external systems or applications, and other subsystems) before granting them usage of sensitive services and access to sensitive data.
    - **Architecture Supports Authentication**
      Claim: The architecture adequately supports the system or subsystem's ability to verify the correctness of the claimed identities of users before granting them usage of sensitive services and access to sensitive data.
    - **Architecture Supports Authorization**
      Claim: The architecture adequately supports the system or subsystem's ability to properly grant and enforce access and usage privileges of authenticated and identified users.
- **Requirements**
  - **Architecture Supports Identification Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived identification requirements that have been allocated to it:
    - "The system shall allow [members of user class X | client application Y] to perform [list of actions Z] before being successfully identified."
    - "The system shall not allow [members of user class X | client application Y] to perform [any | list of actions Z] before being successfully identified."
    - "When under attack, the system shall [detect | prevent] the use of forged identification data."
    - "When the system detects the use of forged identification data, then the system shall [list of actions X]."
    - "The system shall [detect | prevent] the reuse of identification data."
    - "The system shall not require [members of user class X | client application Y] to be reidentified multiple times during a single session (i.e., single sign on)."
    - "The system shall reidentify [members of user class X | client application Y] under [list of conditions]."
    - "The system shall only provide the following feedback to [members of user class X | client application Y] during and as a result of identification."
    - "The data center shall identify all personnel before allowing them to enter."
    - "The name of the employee in the official human resource and payroll databases shall exactly match the name printed on the employee's social security card."

Rationale: This is an official requirement of the United States Social Security Administration.

  - "The system shall support [list of identification mechanisms]."
    Note: This is a security constraint rather than a normal requirement.
  - "The system shall identify [user class X] according to [list of identification processes]."
    Note: This is also a security constraint.

- **Architecture Supports Authentication Requirements**

  Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived authentication requirements that have been allocated to it:

  - "The system shall allow [member of user class X | client application Y] to perform [list of actions Z] before being successfully authenticated."
  - "The system shall not allow [member of user class X | client application Y] to perform [any actions | list of actions Y] before being successfully authenticated."
  - "When under attack, the system shall [detect | prevent] the use of any forged authentication data."
  - "When the system detects the use of forged authentication data, then the system shall [list of actions X]."
  - "The system shall [detect | prevent] the reuse of authentication data."
  - "The system shall reauthenticate [member of user class X | client application Y] under [list of conditions]."
  - "The system shall only provide the following feedback to [member of user class X | client application Y] during and as a result of authentication."
  - "The system shall authenticate all of its users before allowing them to update their user information."
  - "The system shall authenticate all of its users before accepting a credit card payment."
  - "The system shall authenticate all of its client applications before allowing them to use its capabilities."
  - "The data center shall verify the identity of all personnel before permitting them to enter."

- **Architecture Supports Authorization Requirements**

  Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived authorization requirements that have been allocated to it:

  - "The system or subsystem shall allow each customer to obtain access to all of his or her own personal account information."
  - "The system or subsystem shall not allow any customer to access any account information of any other customer."
  - "The system or subsystem shall not allow customer service agents to access the credit card information of customers."
  - "The system or subsystem shall allow customer service agents to automatically email a new customer password to that customer's email address."
    Note: This authorization requirement is questionable because it contains an implied authentication constraint—the use of passwords as opposed other authentication mechanisms such as digital signatures.

- "The system or subsystem shall not allow customer service agents to access either the original or new customer password when emailing the new customer password to the customer's email address."
- "The system or subsystem shall limit remote users to the following services: [list of services]."
- "The system or subsystem shall not allow one or more users to successfully use a denial of service (DoS) attack to flood it with legitimate requests of service."

E.3.1.2 Example Access Control Arguments

The example access control security case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Access Control Subsystem**
  Architectural Decision: The system or subsystem architecture includes a subsystem consisting of a COTS access control application.
  Rationale: This access control application performs the identification, authentication, and authorization of users. It utilizes user identifiers, passwords, biometrics, digital signatures, and more to identify and authenticate users of the system or subsystem being assessed.

E.3.1.3 Example Access Control Evidence

The example access control security case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Configuration Diagram**
  The configuration diagram provides evidence for the *existence* of the access control subsystem by clearly showing the decomposition of the aggregate system or subsystem into its component subsystems, one of which is identified as the access control subsystem.
- **Allocation Diagram**
  The allocation diagram provides evidence for the *location* of the access control subsystem by clearly showing the allocation of the access control software subsystem to a specific hardware server computer.
- **Network Diagrams**
  The network diagram provides evidence for the *network connectivity* of the access control subsystem to the servers hosting the services the access control subsystem protects, the storage devices hosting the sensitive data the access control subsystem protects, and related networks and network devices. This allows the architects to show how the network connectivity of the access control subsystem enables it to protect the associated services and sensitive data.
- **Activity and Collaboration Diagrams**
  The activity and collaboration diagrams provide evidence of the *interactions* between the access control subsystem, other subsystems, users, and external systems. This allows

the architects to show how the interactions involving the access control subsystem enable it to protect the associated services and sensitive data.

- **Access Control COTS Trade Study**
  The access control trade study provides evidence in the form of an evaluation of the various COTS access control products, their vendors, and the relative ability of these products to meet the access control requirements of the system or subsystem. This allows the architects to demonstrate the adequacy of the chosen product to meet its allocated identification, authentication, and authorization requirements.

- **Vendor-Supplied Technical Documentation**
  The vendor-supplied technical documentation for the selected access control product provides evidence that the product has the capabilities needed to meet the access control requirements of the system or subsystem.

- **Security White Paper**
  The access control section of the security white paper supplied by the architects provides evidence in the form of a *description of the capabilities* of the chosen access control product and the results of an *analysis* of the adequacy of this product. This allows the architects to demonstrate the adequacy of the chosen product to meet its allocated identification, authentication, and authorization requirements.

E.3.2 Example Integrity Security Case

Integrity is the degree to which communications or data, hardware, or software components are protected from intentional corruption (e.g., via unauthorized creation, modification, deletion, or replay). As illustrated in Figure 31, because integrity is a subtype of the defensibility quality subfactors malicious harm and protection and because security is a subtype of defensibility, integrity is a quality subfactor of the quality factor security.

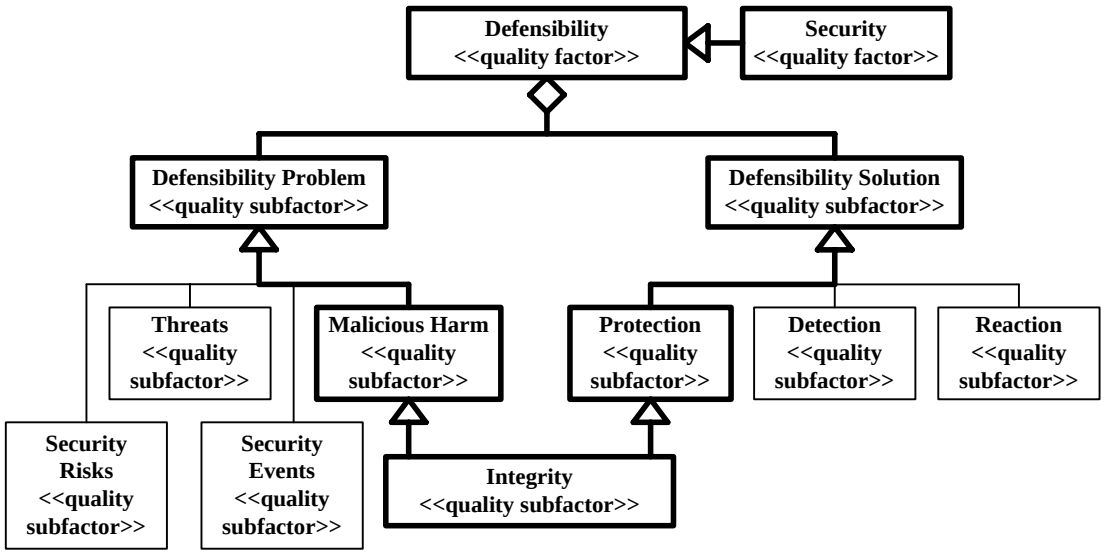

Figure 31: Integrity as a Quality Subfactor of Security

Figure 32 is an example quality case diagram summarizing the claims, arguments, and evidence composing the example integrity security case.

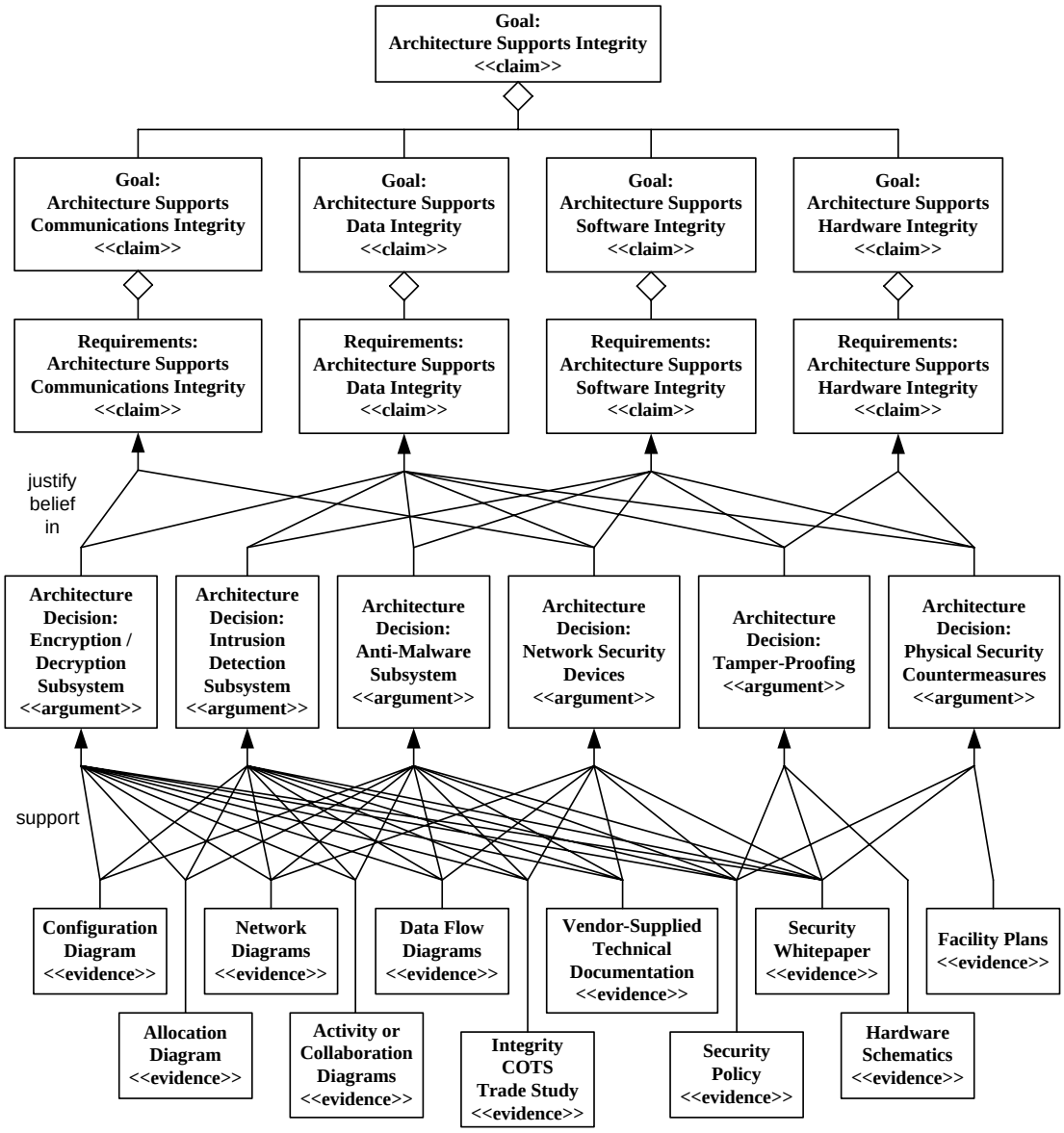

Figure 32: Example Integrity Security Quality Case Diagram

The example integrity security case is made up of the claims, arguments, and evidence presented in Sections E.3.2.1, E.3.2.2, and E.3.2.3, respectively.

E.3.2.1 Example Integrity Claims

The example integrity security case includes the following example claims:

- **Goals**
  - **Architecture Supports Integrity**
    Claim: The architecture adequately supports the system or subsystem's ability to protect communications, data, hardware, and software from intentional corruption (e.g., via unauthorized creation, modification, deletion, or replay).

- *Architecture Supports Communications Integrity*
  Claim: The architecture adequately supports the system or subsystem's ability to ensure the integrity of communications by protecting them from intentional corruption (e.g., via unauthorized creation, modification, deletion, or replay).
- *Architecture Supports Data Integrity*
  Claim: The architecture adequately supports the system or subsystem's ability to ensure the integrity of stored data by protecting it from intentional corruption (e.g., via unauthorized creation, modification, deletion, or duplication).
- *Architecture Supports Hardware Integrity*
  Claim: The architecture adequately supports the system or subsystem's ability to ensure the integrity of hardware by protecting it from intentional corruption (e.g., via unauthorized creation, modification, destruction, or theft).
- *Architecture Supports Software Integrity*
  Claim: The architecture adequately supports the system or subsystem's ability to ensure the integrity of software by protecting it from intentional corruption (e.g., via unauthorized creation, modification, deletion, or theft) by attackers or malware (i.e., malicious software including computer viruses, worms, Trojan horses, adware, and spyware).

- **Requirements**
  - **Architecture Supports Communications Integrity Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived communications integrity requirements that have been allocated to it:
    - *Data Transmitted Integrity Protection Requirement*
      "The system or subsystem shall protect the data it transmits from [sophistication level] attack involving unauthorized addition, modification, deletion, or replay when it transmits the data during the execution of [a set of interactions/use cases] listed in [specified table]."
    - *Data Received Integrity Detection Requirement*
      "The system or subsystem shall determine if communicated data it receives has been modified, if additional data has been added to it, if some protected data has been deleted, and if any protected data has been replayed during execution of [a set of interactions/use cases] when subject to [sophistication level] attack as indicated in [specified table]."
    - *Data Received Integrity Response Requirement*
      "The system or subsystem shall perform [list of application-specific actions] within [time limit] if communicated data it receives has been modified, if additional data has been added to it, if some protected data has been deleted, and if any protected data has been replayed during execution of [a set of interactions / use cases] when subject to [sophistication level] attack as indicated in [specified table]."
  - **Architecture Supports Data Integrity Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived data integrity requirements that have been allocated to it:
    - *Data Stored Integrity Protection Requirement*
      "At least [a percentage such as 99.99%] of the time, the system or subsystem shall

protect the data in [specified table] it stores from unauthorized addition, modification, or deletion by [attacker with specified profile | malware of specified type]."

  - *Data Stored Integrity Detection Requirement*
    "At least [a percentage such as 99.99%] of the time, the system or subsystem shall detect the unauthorized addition, modification, or deletion the data in [specified table] it stores."

- **Architecture Supports Hardware Integrity Requirements**
  Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived hardware integrity requirements that have been allocated to it:
  - *Hardware Integrity Protection Requirement*
    "At least [a percentage such as 99% of the time, the X hardware component shall protect itself from unauthorized addition, modification, destruction, or theft from an attacker having [attacker profile] using [list of attack techniques] for no longer than [time duration]."
  - *Hardware Integrity Detection Requirement*
    "The X hardware device shall be constructed so that successful tampering is easily detectable."

- **Architecture Supports Software Integrity Requirements**
  Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived software integrity (e.g., immunity) requirements that have been allocated to it:
  - *Scanning Requirement*
    "The system or subsystem shall scan all entered or downloaded data and software against the published definitions of known malware."
  - *Currency Requirement*
    "The system or subsystem shall daily update its list of published definitions of known malware."
  - *Disinfection Requirement*
    "Where practical, the system or subsystem shall disinfect any data or software found to contain malware."
  - *Deletion [Quarantine] Requirement*
    "The system or subsystem shall delete [quarantine] all infected data and software that it cannot successfully disinfect."
  - *Notification Requirement*
    "The system or subsystem shall notify a member of the security team if it detects a harmful program during scanning."

E.3.2.2 Example Integrity Arguments

The example integrity security case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Anti-Malware Subsystem**
  Architectural Decision: The system or subsystem architecture includes a subsystem consisting of a COTS antivirus application.

Rationale: The COTS antivirus application supports the achievement of the system or subsystem's communications, data, and software integrity goals and requirements because it detects and quarantines malware (e.g., viruses, worms, and Trojan horses), thereby preventing them from corrupting communications, data, and software.

- **Encryption/Decryption Subsystem**
  Architectural Decision: The system or subsystem architecture includes a subsystem consisting of a COTS encryption/decryption application.

  Rationale: The COTS encryption/decryption application supports the achievement of the system or subsystem's communications and data integrity goals and requirements because it
  - encrypts and decrypts all sensitive communications and stored data
  - incorporates hash codes that are changed if the communications are corrupted

- **Intrusion Detection Subsystem**
  Architectural Decision: The system or subsystem architecture includes a subsystem consisting of a COTS intrusion detection application.

  Rationale: The COTS intrusion detection application supports the achievement of the system or subsystem's data and software integrity goals and requirements because it detects, records, and warns of specific types of intrusions that could result in data and software corruption.

- **Network Security Devices**
  Architectural Decision: The system or subsystem architecture includes multiple COTS network security devices including
  - multiple firewalls that create protected demilitarized zones
  - routers that can be properly configured

  Rationale: The COTS network security devices support the achievement of the subsystem's data and software integrity goals and requirements because they protect sensitive communications, data, and software from corruption.

- **Tamper-Proofing**
  Architectural Decision: The system or subsystem architecture includes the following tamper-proofing countermeasures:
  - volatile random access memory for the temporary storage of all unencrypted data combined with a means to cut power to the device
  - explosive/inflammatory charges to destroy stored data if physical security is compromised (e.g., classified data stored in military aircraft that is shot or forced down)

  Rationale: The system or subsystem architecture supports the achievement of data, hardware, and software integrity goals and requirements because these tamper-proofing countermeasures make it difficult to
  - corrupt data, hardware, and software components
  - not have any such corruptions detected

- **Physical Security Countermeasures**
  Architectural Decision: The system or subsystem architecture includes the following physical security countermeasures (e.g., fences, armed guards, locked doors, secured rooms and vaults, cameras, and sensors):
  Rationale: These physical security countermeasures support the achievement of hardware integrity goals and requirements because they minimize attacker physical access to data, software, and hardware assets.

E.3.2.3 Example Integrity Evidence

The example integrity security case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Configuration Diagram**
  The configuration diagram provides evidence for the *existence* of the anti-malware, encryption/decryption, and intrusion detection subsystems by clearly showing the decomposition of the aggregate system or subsystem into its component subsystems, of which these are three.
- **Allocation Diagram**
  The allocation diagram provides evidence for the *location* of the anti-malware, encryption/decryption, and intrusion detection subsystems by clearly showing the allocation of these software subsystems to specific hardware server computers.
- **Network Diagrams**
  Network diagrams provide evidence for the *network connectivity* of the anti-malware, encryption/decryption, and intrusion detection subsystems as well as the network security devices (e.g., firewalls, routers) to
  - computers and devices involved in sensitive communications
  - disk and tape libraries storing sensitive data
  - servers hosting sensitive software

  This allows the architects to show how the network enables the system or subsystem to ensure the integrity of sensitive communications, data, and software.
- **Activity and Collaboration Diagrams**
  The activity and collaboration diagrams provide evidence of the *interactions* between the anti-malware, encryption/decryption, and intrusion detection subsystems, other subsystems, users, and external systems. This allows the architects to show how these interactions enable the system or subsystem to protect the integrity of its sensitive communications, data, and software.
- **Integrity COTS Trade Study**
  The integrity trade study provides evidence in the form of an evaluation of the various COTS anti-malware, encryption/decryption, and intrusion detection products, network security devices, their vendors, and the relative ability of these products to meet the integrity requirements allocated to the system or subsystem. This allows the architects to

demonstrate the adequacy of the chosen products to meet their allocated integrity requirements.

- **Vendor-Supplied Technical Documentation**
  The vendor-supplied technical documentation for the selected anti-malware product, encryption/decryption product, intrusion detection product, and network security devices provide evidence that these products has the capabilities needed to meet the integrity requirements of the system or subsystem.

- **Security Policy**
  The security policy produced by the security team provides evidence in the form of countermeasures used to ensure integrity of data, hardware, and software components.

- **Security White Paper**
  The relevant sections of the security white paper provide evidence in the form of *descriptions of the capabilities* of the chosen products and the results of *analyses* of the adequacy of these products. This allows the architects to demonstrate the adequacy of the chosen products to meet their allocated integrity requirements.

- **Hardware Schematics**
  The hardware schematics provide evidence of tamper-proofing of sensitive hardware components.

- **Facility Plans**
  The facility plans provide evidence of physical security countermeasures such as the location of secure rooms and vaults, the location of armed guards and fences, and the location of security devices such as cameras and door-locking mechanisms.

E.3.3 Example Privacy Security Case

Privacy is the degree to a system or subsystem keeps sensitive identifications, data, and communications secret from unauthorized individuals, organizations, software applications, and other systems. Privacy consists of the following security subfactors:

- **Anonymity** is the degree to which a system or subsystem keeps private the identity of individuals, organizations, applications, and systems from unauthorized individuals, organizations, software applications, or systems.

- **Confidentiality** is the degree to which a system or subsystem keeps private sensitive data and communications from unauthorized individuals, organizations, software applications, or systems.

Figure 33 is an example quality case diagram summarizing the claims, arguments, and evidence composing the example privacy security case.

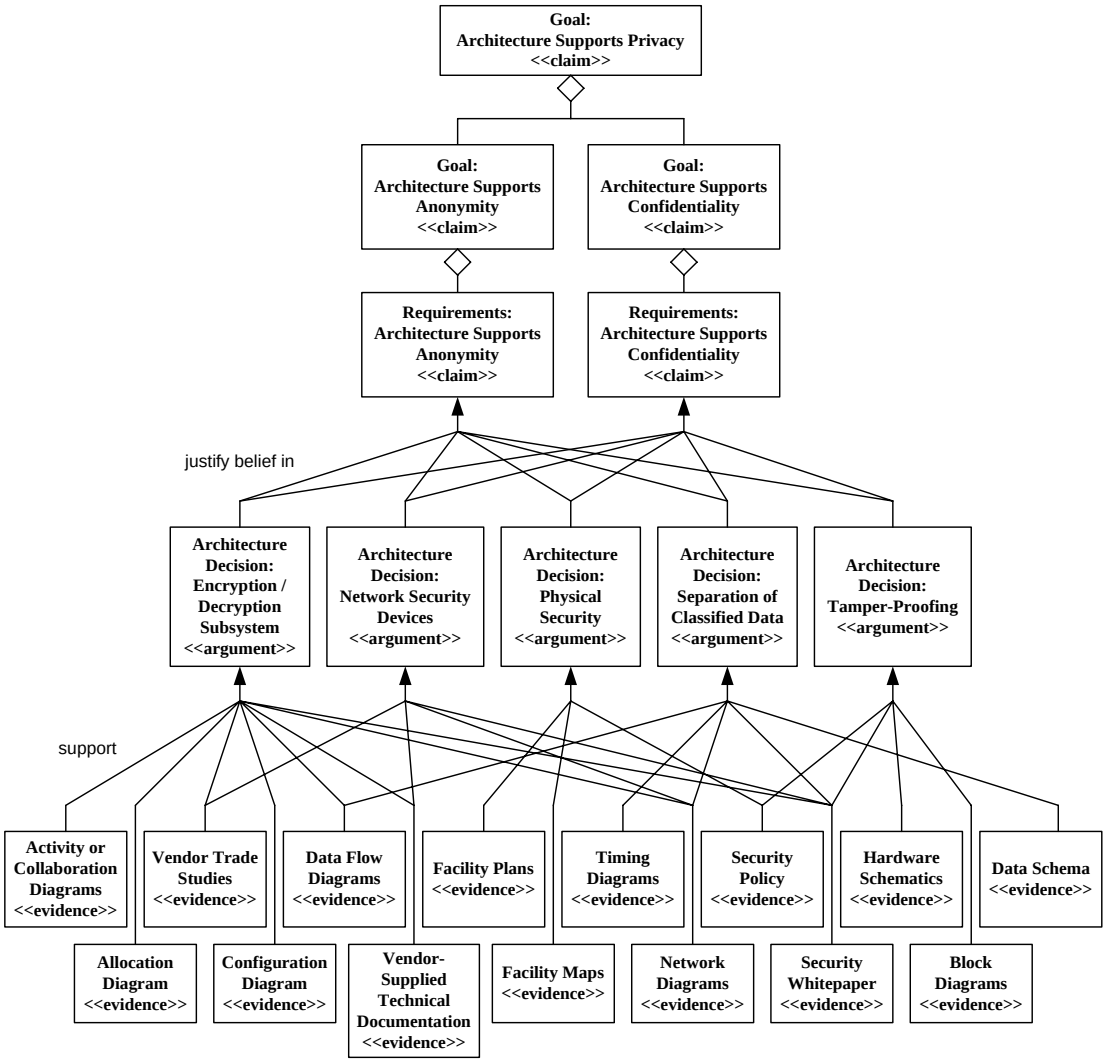

Figure 33: Example Privacy Quality Case Diagram

The example privacy security case is made up of the claims, arguments, and evidence presented in Sections E.3.3.1, E.3.3.2, and E.3.3.3, respectively.

E.3.3.1 Example Privacy Security Claims

The example privacy security case includes the following example claims:

- **Privacy Requirements**
  Privacy is the degree to which sensitive identifications, data, and communications is kept secret from unauthorized individuals, organizations, software applications, and systems:[68]
  - **Anonymity (identity) Requirements**
    whereby anonymity protects identities
    - "The system or subsystem shall prevent the unauthorized disclosure of the identity of [some class of users]."
  - **Confidentiality Requirements**
    whereby confidentiality protects data and communications
    - "The system or subsystem shall prevent the unauthorized disclosure of [some class of sensitive data or communications]."

E.3.3.2 Example Privacy Arguments

The example privacy security case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Encryption/Decryption Subsystem**
  Architectural Decision: The system or subsystem architecture will include a subsystem consisting of a COTS encryption/decryption application.

  Rationale: This subsystem supports confidentiality by encrypting and decrypting all sensitive data when both stored and transmitted.

- **Network Security Devices**
  Architectural Decision: The system or subsystem architecture incorporates network security devices including firewalls and routers. For example, the architecture incorporates multiple firewalls at strategic locations in the network.

  Rationale: The firewalls create "demilitarized zones" that protect sensitive data from unauthorized external access.

- **Separation of Classified Data**
  Architectural Decision: The system or subsystem architecture separates classified data at different levels of classification onto separate physical processors, OS processes, and storage (e.g., disk and tape libraries).

---

[68] Technically, security is concerned with preventing and detecting the *malicious* disclosure of sensitive identities, data, and communications during an attack, whereas safety is concerned with preventing and detecting their *accidental* disclosure during an accident. Note that both attacks and accidents can result in the same harm to valuable assets (i.e., to individuals, organizations, and their sensitive data) and that the architects can give the same arguments as part of both safety and security cases because they can use the same architecture mechanisms (controls) as both safeguards and countermeasures.

Rationale: This separation makes it difficult for data and services at higher levels of classification to mix with and be accessed by data and services at lower levels of classification.

- **Physical Security**
  Architectural Decision: The system architecture places its valuable assets (e.g., people and property such as sensitive data, valuable hardware, and private services) within locked facilities and vaults protected by cameras and armed guards.

  Rationale: The architecture of the system or subsystem supports the achievement of its allocated physical security requirements.

- **Tamper-Proofing**
  Architectural Decision: The architecture of the system or subsystem supports the achievement of its data confidentiality requirements by using
  - volatile random access memory for the temporary storage of all unencrypted data combined with a means to cut power to the device
  - explosive/inflammatory charges to destroy stored data if physical security is compromised (e.g., classified data stored in military aircraft that is shot or forced down).

E.3.3.3 Example Privacy Evidence

The example privacy security case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Activity/Collaboration Diagram**
  One or more activity and collaboration diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    Activity and collaboration diagrams document the existence, location, and use of the encryption/decryption subsystem including interactions between the encryption/decryption subsystem, other subsystems, users, and external systems.
- **Allocation Diagram**
  One or more allocation diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    An allocation diagram documents the existence and location of the encryption/decryption subsystem by clearly showing the allocation of the encryption/decryption subsystem to hardware (i.e., one or more server computers).
- **Block Diagrams**
  One or more block diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Tamper-Proofing
    Block diagrams provide evidence for adequate tamper-proofing by documenting the existence of explosive/inflammatory charges to destroy stored data if physical secu-

rity is compromised (e.g., classified data stored in military aircraft that is shot or forced down).

- **Configuration Diagram**
  One or more configuration diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    A configuration diagram documents the existence of the encryption/decryption subsystem by clearly documenting the decomposition of the aggregate system or subsystem into its component subsystems with the encryption/decryption subsystem identified.
- **Data Flow Diagram**
  One or more data flow diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    Data flow diagrams document the existence, location, and use of the encryption/decryption subsystem by documenting the movement of encrypted and decrypted data through the system or subsystem.
  - Separation of Classified Data
    Data flow diagrams document the proper separation of sensitive data at different levels of classification by documenting the movement of sensitive data of different classification levels through the system or subsystem.
- **Data Schemas**
  One or more logical and physical data schemas are used as valid evidence to support the architects' privacy arguments as follows:
  - Separation of Classified Data
    Logical and physical data schemas document the proper separation of sensitive data at different levels of classification by documenting the storage of data of different classification levels.
- **Facility Plans**
  The facility floor plans are used as valid evidence to support the architects' privacy arguments as follows:
  - Physical Security
    Facility floor plans provide acceptable evidence for adequate physical security by documenting the
    - location of valuable assets including people, property (e.g., sensitive data, hardware, and software), and access to service
    - location of armed guards, locked doors, and secure areas (e.g., rooms and vaults)
    - location of security devices such as cameras, motion sensors, and physical access devices (e.g., locks, biometric sensors)
  - Facility Maps
    Facility maps are used as valid evidence to support the architects' privacy arguments as follows:

- Physical Security
  One or more facility maps document physical security by documenting the location of security fences and sensors.

- **Hardware Schematics**
  One or more hardware schematics are used as valid evidence to support the architects' privacy arguments as follows:
  - Tamper-Proofing
    Hardware schematics provide evidence for adequate tamper-proofing by documenting the existence of explosive/inflammatory charges to destroy stored data if physical security is compromised (e.g., classified data stored in military aircraft that is shot or forced down).

- **Network Diagram**
  One or more network diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    Network diagrams document the existence and location of the encryption/decryption subsystem by showing the connectivity between the server(s) hosting the encryption/decryption subsystem, the servers hosting and storage devices storing the data the encryption/decryption subsystem protects, and related networks and network devices, enabling the architects to show how the location of the encryption/decryption subsystem enables it to protect the associated sensitive data.
  - Network Security Devices
    Network diagrams document the network devices having security functions including the networks, clients, and servers, enabling the architects to show
    - the existence, location, and adequacy of the network devices having security functions (e.g., firewalls and routers)
    - associated demilitarized zones
    - potential attack paths
    - how the location of these devices enables them to properly perform their security functions
  - Separation of Classified Data
    Network diagrams document the proper separation of sensitive data at different levels of classification by showing connectivity between the server(s) hosting the software components creating/accessing/using/storing sensitive data, storage devices (e.g., disk and tape libraries) that store such data, and the possible paths along which such data can flow.

- **Security Policy**
  The security policy used as valid evidence to support the architects' privacy arguments as follows:
  - Physical Security
    The security policy documents how physical security will be achieved.
  - Tamper-Proofing
    The security policy documents how adequate tamper-proofing will be achieved.

- **Security White Paper**
  Security white papers are used as valid evidence to support the architects' privacy arguments as follows:
  - Encryption/Decryption Subsystem
    The security white paper documents the existence, location, capabilities, and use of the encryption/decryption subsystem in its encryption/decryption section.
  - Network Security Devices
    The security white paper documents the existence, location, capabilities, and use of the network devices having security functions (e.g., firewalls and routers) in its network security section.
  - Separation of Classified Data
    The security white paper documents the proper separation of sensitive data at different levels of classification by documenting how data of different classification levels are kept separate.
  - Tamper-Proofing
    The security white paper on tamper-proofing provides evidence for adequate tamper-proofing by documenting the use of volatile random access memory for the temporary storage of all unencrypted data combined with a means to cut power to it when necessary so that the data is deleted.
- **Timing Diagrams**
  One or more timing diagrams are used as valid evidence to support the architects' privacy arguments as follows:
  - Separation of Classified Data
    Timing (and process) diagrams document the proper separation of sensitive data at different levels of classification by documenting the allocation of data at different classification levels to different OS processes or time slices.
  - Vendor-Supplied Technical Documentation
    Vendor-supplied technical documentation is used as valid evidence to support the architects' privacy arguments as follows:
    - Encryption/Decryption Subsystem
      The vendor-supplied technical documentation for the selected encryption/decryption subsystem documents the capabilities of this subsystem in the associated sections.
    - Network Security Devices
      The vendor-supplied technical documentation for the selected network devices having security functions (e.g., firewalls and routers) documents the capabilities of these devices in the associated sections.
  - Vendor Trade Studies
    Vendor trade studies are used as valid evidence to support the architects' privacy arguments as follows:
    - Encryption/Decryption Subsystem
      The encryption/decryption vendor trade study documents the capabilities of the selected subsystem in the section on the selected subsystem.

– Network Security Devices
The security vendor trade study documents the security capabilities of the network devices having security functions (e.g., firewalls and routers).

E.4.4 Example Security Auditing Security Case

The security subfactor, *security auditing*, is the degree to which the system or subsystem enables security personnel to audit the status and use of security mechanisms[69] by analyzing security-related events.

The example security auditing security case is made up of the claims, arguments, and evidence presented in Sections E.3.4.1, E.3.4.2, and E.3.4.3, respectively.

E.4.4.1 Example Security Auditing Claims

The example security auditing security case includes the following example claims:

- **Security Auditing Requirements**
whereby security auditing is the degree to which the system or subsystem enables security personnel to audit the status and use of security mechanisms by analyzing associated security-related events
  - **Security Audit Control Requirements**
    - "At least 99.9% of the time, the system or subsystem shall automatically start security auditing within 0.1 seconds of startup and restart."
    - "At least 99.99% of the time, the system or subsystem shall enable [an explicitly identified and authenticated list of individuals, user roles, or user groups] to start and stop security auditing."
    - "At least 99.99% of the time, the system or subsystem shall enable [an explicitly identified and authenticated list of individuals, user roles, or user groups] to read and modify the security events to be audited."
  - **Security Audit Log Content Requirements**
    - "The system or subsystem shall record at least 99.99% of the following security-related events:
      - security audit startup
      - security audit shutdown
      - access control events (including successful and unsuccessful identification, authentication, and authorization events)
      - changes in access control
      - intrusion detection (including attempts to control security auditing or to modify the security log)
      - [application-specific entries]"

[69] See http://www.opfro.org/Glossary/GlossaryS.html#security_mechanism for more information.

- "At least 99.99% of the time, the system or subsystem shall include the following information within each security audit record:
  - date and time of the security event
  - type of security event
  - parties (e.g., human, external application, software process) to the security event
  - outcome (e.g., success, failure) of security event
  - [application-specific entries]"

- **Security Audit Reporting**
  - "At least 99.9% of the time, the system or subsystem shall enable the following explicitly identified and authenticated individuals, user roles, or user groups to read/search/sort the security audit records: [an application-specific list]."
  - "At least 99.99% of the time, the system or subsystem shall enable the following explicitly identified and authenticated individuals, user roles, or user groups to read/generate the security audit reports: [an application-specific list]."

- **Security Audit Log Protection**
  - "At least 99.9% of the time, the system or subsystem shall protect the audit log contents from being modified for at least one day when under attack by a hacker of medium-sophistication."
  - "At least 99.9% of the time, the system or subsystem shall detect any attempts to modify the audit log contents by a hacker of medium-sophistication during a one day period."
  - "At least 99.9% of the time, the system or subsystem shall protect the audit log contents for at least one day from unauthorized deletion by a hacker of medium sophistication."
  - "At least 99.99% of the time, the system or subsystem shall notify the following identified and authenticated individuals, user roles, or user groups if the security audit log exceeds [an application-specific size]: [an application-specific list]."
  - "At least 99.99% of the time, the system or subsystem shall retain at least the 500 most recent audit log records when any of the following exceptional conditions occur:
    - failure of audit hardware or software components
    - exhaustion of audit log storage"

E.3.4.2 Example Security Audit Security Case Arguments

The example security audit security case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **COTS Security Audit Subsystem**
  Architectural Decision: The architecture includes a COTS security audit subsystem.

  Rationale: Compared to an in-house developed system, the COTS subsystem is of higher quality, costs significantly less, and will decrease development time. It meets the security audit control, audit log contents, audit reporting, and audit log protection requirements.

E.3.4.3 Example Security Audit Security Case Evidence

The example security audit security case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Configuration Diagram**
  A configuration diagram clearly shows the decomposition of the aggregate system or subsystem into its component subsystems with the security audit subsystem identified.
- **Allocation Diagram**
  An allocation diagram clearly shows the allocation of the security audit subsystem to hardware (i.e., one or more server computers).
- **Network Diagram**
  One or more network diagrams show connectivity between the server(s) hosting the security audit subsystem, other servers, and related networks and network devices (e.g., firewalls and routers), enabling the architects to show
  - where security event information is generated
  - where security events are logged
  - the flow of security event notifications
  - how the location of the security audit subsystem enables it to properly perform its functions
- **Collaboration Diagram and/or Activity Diagram**
  One or more collaboration diagrams or activity diagrams document interactions between the security audit subsystem, other subsystems, and attackers.
- **Security White Paper**
  A security white paper documents the capabilities of the security audit subsystem.
- **Trade Study**
  A trade study documents the capabilities of the security audit subsystems that were considered for use in the system architecture.
- **Vendor-Supplied Technical Description**
  Vendor-supplied technical description documents the capabilities of the security audit subsystem.

E.4 Example Stability Case

Stability is the degree to which a system or subsystem continues to deliver *mission-critical* services during a given time period under a given operational profile regardless of any failures whereby the

- failures may prevent the system or subsystem from delivering less critical services
- failures limiting the delivery of mission-critical services occur at unpredictable times
- root causes of such failures are difficult to identify efficiently

A system is stable to the degree that minor failures do not cascade into major failures.

E.4.1 Example Stability Protection Case

As a stability subfactor, *stability protection* is the degree to which the system is stable as opposed to detecting when it is not stable or reacting properly when it is not stable.

Figure 34 is an example stability quality case diagram summarizing the claims, arguments, and evidence composing the example stability case.

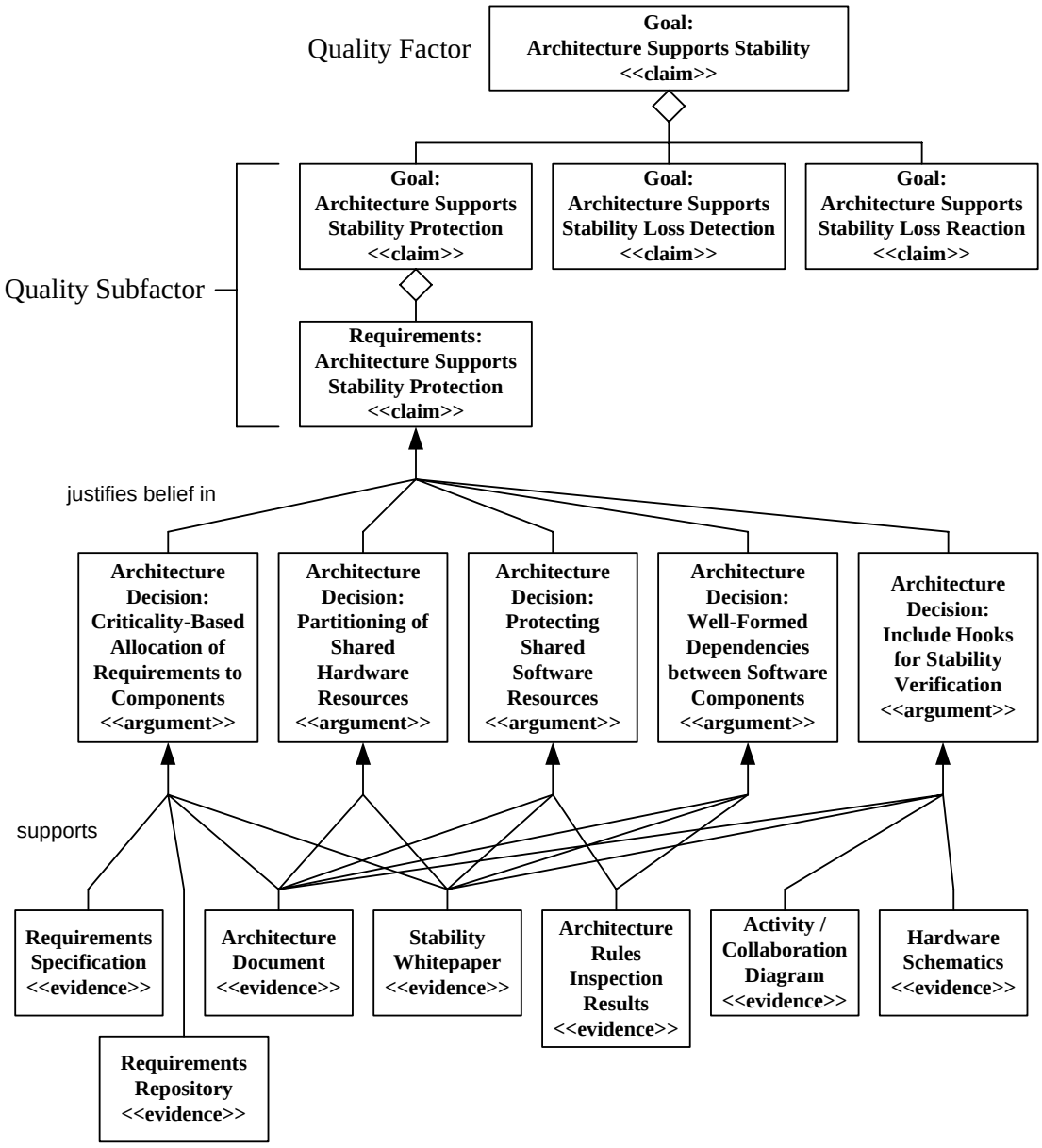

Figure 34: Example Stability Quality Case Diagram

The example stability quality case is made up of the claims, arguments, and evidence presented in Sections E.4.1.1, E.4.1.2, and E.4.1.3, respectively.

E.4.1.1 Stability Claims

The example stability performance case includes the following example claims:

- **Goals**
  - **Architecture Supports Stability Goal**
    Claim: The system or subsystem architecture adequately supports the stability of the system or subsystem.

- **Requirements**
  - **Architecture Supports Stability Requirements**
    Claim: The architecture adequately supports the system or subsystem's ability to meet the following derived stability requirements that have been allocated to it:
    - *Mean Time Between Critical Failures (Stability Requirement)*
      "Under normal operating conditions, the system shall not lose mission-critical functionality more often than an average of once every 5,000 hours of operation (i.e., the mean time between critical failures[70] is at least 5,000 hours of operation)."
    - *Cascading Failures (Stability Requirement*)
      "Under normal operating conditions, the system shall prevent critical failures in any lower criticality subsystem from cascading into failures of its higher criticality subsystems with a mean time between critical failures of at least 5,000 hours of operation."
    - *Well-Formed Interactions (Stability Constraint)*
      "Interactions between subsystems at different levels of criticality shall be well formed."

E.4.1.2 Example Stability Arguments

The example stability case includes the following example arguments covering the architectural decisions the architects have made to justify the assessors' belief in the associated claims:

- **Critically Based Allocation of Requirements to Components**
  Architectural Decision: Except in a small number of identified and justified cases, each architectural component implements requirements that are classified as having a single criticality level.

  Rationale: System requirements are classified into a partially order set of distinct criticality levels according to overall mission needs. Architectural components at higher levels of criticality must be better engineered than components at lower levels of criticality. Allocating a mixture of critical and non-critical requirements to a single architectural component would require that the component be engineered at the highest critical level of all requirements allocated to the component, which would significantly increase the effort and cost required to design, implement, and test the component.

  This architecture decision improves stability because it lowers engineering costs and lowers the coupling between software that implements higher criticality and lower criticality requirements.

[70] A critical failure is defined as any failure that results in the loss of mission-critical functionality.

- **Partitioning of Shared Hardware Resources**
  Architectural Decision: Based on the criticality of the requirements allocated to software architectural components, the architecture partitions and allocates the following kinds of shared hardware resources to these software components:
  - CPU cycles
  - communication bandwidth
  - storage

  Rationale: Hardware resource budgets are established for each software component and performance analyses of current estimated software component resource utilization verify that the allocated budgets should be adequate.[71] Partitioning hardware resources is the simplest approach to ensure critical services will have adequate hardware resources. Lower criticality software components must not overuse storage because they may corrupt code or data of higher level criticality components.

  This architectural decision improves stability because it prevents the overuse of hardware resources by lower criticality software components from causing the failure of higher criticality software components.

- **Protecting Shared Software Resources**
  Architectural Decision: Designers and implementers are required to adhere to the following architectural rules:
  - Software components are forbidden to use system calls (e.g., `kill`) that can interfere with the execution of other applications. Only system managers can terminate ill-behaving software components.
  - The longest system call must be within the designated limits, which are published and available for schedulability analysis.
  - All shared libraries used by software components are re-entrant.

  Rationale: The system architecture partitions shared software resources to prevent software components at different levels of criticality from corrupting shared services (e.g., OS and middleware) by ensuring that the software components cannot interfere with each other via those shared services. If lower criticality software components takes too long when using a shared resource, it may delay the execution of a higher criticality software component that is waiting for the service. If the software in a shared library is not re-entrant, then the next user could be adversely affected if the calling software component fails in the middle of a library call.

  This architecture decision improves stability because it prevents failures of lower criticality components from causing failures of higher criticality components.

[71] Note that at the current time during development, the adequacy of the allocated resource budgets cannot be completely verified because the detailed design of the subsystem is not yet complete.

- **Well-Formed Dependencies between Software Components**
  Architectural Decision: Designers and implementers are required to adhere to the following architectural rules:
  - Software components allocated requirements of different criticality can only interact via asynchronous message passing.
  - The logical dependencies resulting from message exchanges between software components that are allocated requirements of different criticality must be verifiably well formed.[72]

  Rationale: If two software components interact via shared variables and one fails while holding a lock on the shared variable, then the other software components needing access to the shared variable will be locked. A similar problem will occur if send-wait (aka, synchronous message passing) is used. Restricting interactions to asynchronous message passing avoids this kind of failure. If all logical dependencies resulting from message exchanges are verified to be well-formed, then faults and failures in the less critical component will not cause failures in the more critical components.

  This architectural decision improves stability because it prevents failures of lower-criticality components from causing failures of higher-criticality components.

- **Include Hooks for Stability Verification**
  Architectural Decision: Hardware and software components include hooks for later stability verification.
  - Hardware hooks include such things as testing pins on the board or a network testing harness.
  - Software hooks include annotations for static analysis, instrumented code, and interfaces for test messages.

  Rationale: Because stability depends on reliability, if the hardware component does not have the required reliability, then it should provide hardware interfaces to support subsequent diagnostics. Such hardware testing access enables testers to trace potential problems back to their source components during integration testing.

  Note that although this architectural decision is usually used during prototyping and development rather than for production, care must be exercised because removal of the hook might have unexpected side-effects to the production system (i.e., you are delivering something other than what you tested). Note that if the hooks are left in, then this architectural decision may have negative security implications. Hooks may also have negative impacts on performance.

[72] By "well-formed dependency" we mean that the critical service is not allowed to use a less critical service unless we can verify that the critical services cannot be compromised by the faults and failures in the less critical services. For example, in a passenger jet, the navigation system may use the entertainment subsystem to inform passengers about the current location and the speed of the plane. However, to do that we must be able to verify that faults and failures in the entertainment system cannot compromise navigation.

This architectural decision improves stability because it enables the early detection and removal of stability problems.

E.4.1.3 Example Stability Evidence

The example stability case includes the following acceptable example evidence that could be supplied by the architects to support their associated arguments:

- **Requirements Specification**
  The system or subsystem requirements specification documents the criticality levels of the individual requirements[73] allocated to the system or subsystem respectively.
- **Requirements Repository**
  The requirements repository stores the criticality levels of the individual requirements as metadata associated with the individual requirements.
- **Architecture Document**
  The architecture document provides an overview of the architecture decisions improving stability and their rationales.
- **Architecture Rule Inspection Results**
  This documentation reports the results of inspections that verify compliance with architectural rules including rules to ensure that
  - hardware resources are properly partitioned
  - interactions between software components are well formed
- **Stability White Paper**
  This document provides detailed descriptions of architecture decisions improving stability and their rationales.
- **Activity/Collaboration Diagram**
  One or more activity or collaboration diagrams document test interfaces (e.g., using UML stereotypes).
- **Hardware Schematics**
  One or more hardware schematics document test interfaces.

[73] This includes classifying the criticality of all functional, data, interface, and quality requirements.

Appendix F Example Contract Language

To ensure that adequate resources are included in architecture plans and schedules, it is important for acquisition organizations to include appropriate contract language in requests for proposals (RFPs) and development contracts. The following is provided as example, tailorable contract language to mandate the performance of system architecture quality assessments based on architect-supplied quality cases:

1. As the system architecture is developed, the contractor shall support incremental assessments of the quality of the system architecture.
2. Support for system architecture quality assessments shall be documented in architecture plans, procedures, and schedules.
3. The system architecture quality assessment method shall be based on architect-developed quality cases provided and presented to the assessment team(s).
4. The system architecture shall be assessed in terms of the ability of the architectures of its subsystems to sufficiently support the achievement of the prioritized quality goals and requirements that have been derived and allocated to the subsystems.
5. The assessment team shall be led by members of the acquisition organization.
6. During these subsystem architecture quality assessments, the subsystem architects shall make compelling quality cases to the assessment team that their architectures sufficiently support the achievement of the associated quality goals and requirements.
7. The subsystem architects shall provide clear and compelling arguments stating the architectural decisions that they have made and their rationales for making these decisions.
8. The subsystem architects shall provide the assessment team with access to sufficient evidentiary documentation (or hold acquirer-witnessed demonstrations) to support their arguments and thereby justify their claims that their architectures adequately support the subsystem's ability to achieve its quality goals and requirements.

The preceding language is intended to be tailored to meet the needs of the specific contract. For example, the following aspects of the above languages should be considered for tailoring:

- Item 2 – Where should support for the system architecture quality assessments be documented? Who approves the adequacy and quality of this documentation?
- Item 5 – Who should lead the assessment team: the acquisition organization or contractor? What organizations are responsible for what aspects of the assessments? To what degree should assessments be independent of the architecture team? Should the assessments be internal to the development organization or performed by an independent or-

ganization? Should the acquisition organization take part in the assessment team? If so, should they lead the assessment team?

- What about subcontractors? Must the prime contractor mandate system architecture quality assessments onto subcontractors of major subsystems? Down to what level? Should the acquisition organization be involved in these assessments?
- What if the acquisition organization and contractor do not agree on the scope of the assessments (e.g., subsystems and quality factors)?

References

URLs are valid as of the publication date of this document.

[Bishop 98] Bishop, Peter & Bloomfield, Robin. *A Methodology for Safety Case Development.* http://www.adelard.co.uk/resources/papers/pdf/sss98web.pdf (1998).

[Clements 02] Clements, Paul; Kazman, Rick; & Klein, Mark. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.

[Firesmith 03] Firesmith, Donald G.. "Using Quality Models to Engineer Quality Requirements." *Journal of Object Technology (JOT) 2*, 5 (September-October 2003): 67-75. http://www.jot.fm/issues/issue_2003_09/column6

[ISO 91] International Organization for Standardization. *International Standard ISO/IEC 9126. Information technology–Software product evaluation–Quality characteristics and guidelines for their use*. Geneva, Switzerland: International Organization for Standardization, International Electrotechnical Commission (IEC), 1991.

[OPFRO 06] Open Process Framework Repository Organization. *OPEN Process Framework Repository Organization Website.* http://www.opfro.org (2006).

[Weinstock 04] Weinstock, Charles B.; Goodenough, John B.; & Hudak, John J. *Dependability Cases* (CMU/SEI-2004-TN-016) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tn016.html

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	July 2006	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
QUASAR: A Method for the Quality Assessment of Software-Intensive System Architectures	FA8721-05-C-0003

6. AUTHOR(S)

Donald Firesmith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2006-HB-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This handbook documents the QUASAR (QUality Assessment of System ARchitectures) method for assessing the quality of the architecture of a software-intensive system. It begins by discussing the challenges that are faced when assessing a system's architecture and out-lines the development history of the method. The next section of the handbook documents the concept of quality cases and the claims, arguments, and evidence that compose them. This is followed by a description of the teams that collaborate to perform QUASAR tasks. Next, individual tasks and associated steps performed as part of the QUASAR method are documented. Next, the work products produced by these teams when performing these tasks are described. Finally, lessons learned during the development and use of the method when assessing the quality of major subsystems during the development of a very large, software-intensive system of systems are presented. Also provided are appendices that define common quality factors and subfactors, offer reusable checklists, and give examples of quality cases. The example quality cases illustrate valid quality goals and requirements that compose claims, example architecture decisions and associated rationales that compose arguments, and the types of evidence that architects might provide.

14. SUBJECT TERMS	15. NUMBER OF PAGES
acquisition, architecture, architecture evaluation, COTS, software-intensive system, metric, interoperability, metric, security, quality, system architecture, safety	266

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL